

UNIVERSITÀ DEGLI STUDI DI NAPOLI “FEDERICO II”

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

Department of Industrial Engineering



Ph.D. Dissertation

XXXI Course

Development of a high-order parallel solver for Direct and Large Eddy simulations of turbulent flows

Enrico Maria De Angelis

Supervisor

Gennaro COPPOLA

Francesco CAPUANO

Coordinator

Michele GRASSI

DECEMBER 2018

Contact information:

Enrico Maria De Angelis

Università degli Studi di Napoli “Federico II”

Department of Industrial Engineering

P.le Tecchio 80, 80125, Napoli

Italy

email: enricomaria.deangelis@unina.it

COLOPHON

This document was created using L^AT_EX 2_ε and its source files were edited by means of the Vim text editor, enhanced by the powerful VimT_EX plugin. The PGFPLOTS package was used plot functions, whereas the sketches where typeset using the TikZ sledgehammer; the compilation overhead due to the use of both packages was overcome through the combined use of the external library of TikZ, the TikZscale package, and a stroke of LuaL^AT_EX, once in a while.

Abstract

Turbulence is inherent in fluid dynamics, in that laminar flows are rather the exception than the rule, hence the longstanding interest in the subject, both within the academic community and the industrial R&D laboratories.

Since 1883, much progress has been made, and statistics applied to turbulence have provided understanding of the scaling laws which are peculiar to several model flows, whereas experiments have given insight on the structure of real-world flows, but, soon enough, numerical approaches to the matter have become the most promising ones, since they lay the ground for the solution of high Reynolds number unsteady Navier-Stokes equations by means of computer systems.

Nevertheless, despite the exponential rise in computational capability over the last few decades, the more computer technology advances, the higher the Reynolds number sought for test-cases of industrial interest: there is a natural tendency to perform simulations as large as possible, a habit that leaves no room for wasting resources. Indeed, as the scale separation grows with Re , the reduction of wall clock times for a high-fidelity solution of desired accuracy becomes increasingly important. To achieve this task, a CFD solver should rely on the use of appropriate physical models, consistent numerical methods to discretize the equations, accurate non-dissipative numerical schemes, efficient algorithms to solve the numerics, and fast routines implementing those algorithms.

Two archetypal approaches to CFD are direct and large-eddy simulation (DNS and LES respectively), which profoundly differ in several aspects but are both “eddy-resolving” methods, meant to resolve the structures of the flow-field with the highest possible accuracy and putting in as little spurious dissipation as possible. These two requirements of accurate resolution of scales, and energy conservation, should be addressed by any numerical method, since they are essential to many real-world fluid flows of

industrial interest. As a consequence, high order numerical schemes, and compact schemes among them, have received much consideration, since they address both goals, at the cost of a lower ease of application of the boundary condition, and a higher computational cost. The latter problem is tackled with parallel computing, which also allows to take advantage of the currently available computer power at the best possible extent.

The research activity conducted by the present author has concerned the development, from scratch, of a three-dimensional, unsteady, incompressible Navier-Stokes parallel solver, which uses an advanced algorithm for the process-wise solution of the linear systems arising from the application of high order compact finite difference schemes, and hinges upon a three-dimensional decomposition of the cartesian computational space.

The code is written in modern Fortran 2003 — plus a few features which are unique to the 2008 standard — and is parallelized through the use of MPI 3.1 standard's advanced routines, as implemented by the OpenMPI library project. The coding was carried out with the objective of creating an original CFD high-order parallel solver which is maintainable and extendable, of course within a well-defined range of possibilities. With this main priority being outlined, particular attention was paid to several key concepts: modularity and readability of the source code and, in turn, its reusability; ease of implementation of virtually any new explicit or implicit finite difference scheme; modern programming style and avoidance of deprecated old legacy Fortran constructs and features, so that the world wide web is a reliable and active means to the quick solution of coding problems arising from the implementation of new modules in the code; last but not least, thorough comments, especially in critical sections of the code, explaining motives and possible expected weak links. Design, production, and documentation of a program from scratch is almost never complete. This is certainly true for the present effort.

The method and the code are verified against the full three-dimensional Lid-Driven Cavity and Taylor-Green Vortex flows. The latter test is used also for the assessment of scalability and parallel efficiency.

Contents

1	Introduction	1
1.1	The CFD Solver	1
1.1.1	An in-lab code	1
1.1.2	Fortran	4
1.1.3	High order compact schemes	6
1.1.4	Parallel computing	7
1.2	Thesis layout	9
2	High accuracy schemes	11
2.1	Need for high resolution characteristics	12
2.1.1	Formal accuracy and high-frequency harmonics . . .	12
2.1.2	A first example: the Padé scheme	13
2.2	A schematic graphical representation	15
2.3	General formulation for the symmetric case	16
2.3.1	Convenient linear operators	17
2.3.2	Explicit symmetric formulæ	17
2.3.3	Compact symmetric formulæ	19
2.4	Spectral analysis	21
2.4.1	Application to compact schemes	23
2.5	Asymmetric compact schemes	25
2.5.1	Interpolation schemes on staggered stencils	25
2.5.2	First differentiation schemes on staggered stencils . .	27
2.5.3	Second differentiation schemes on collocated stencils	28
2.6	The algorithm to compute the coefficients	30
2.7	Hermitian compact schemes	33
2.8	Conservative compact schemes	33
2.9	Further considerations	34
2.9.1	Ill-posed schemes	34

2.9.2	Parametric schemes	36
2.9.3	The symmetry and the order of accuracy	36
2.9.4	CCD and other schemes	38
2.9.5	Compact schemes as linear combinations of explicit finite differences	38
2.10	Impact of boundary schemes over global accuracy	40
3	Parallelization of linear banded systems	45
3.1	Exact parallelization	46
3.1.1	Two processors	46
3.1.2	Multi processors	49
3.2	The truncated SPIKE algorithm	55
3.3	Other approximate parallelizations	56
4	Code walk-through	59
4.1	Foreword	59
4.2	Need for Git	60
4.3	Input/output	61
4.3.1	Example input file	61
4.3.2	Parallel-independent output	63
4.4	Layout of variables and array indexing	64
4.4.1	Indexing of the grids	65
4.5	The Cartesian grid of MPI processes	69
4.6	Process-local array indexing	70
4.7	The Blk3D class	71
4.8	Compact operators	75
4.8.1	Selection of finite difference schemes from input . . .	77
4.8.2	Storage of banded matrices	79
4.8.3	Distribution of banded matrices	81
4.9	The count of communications	82
4.9.1	Count for slab- and pencil-based solvers	82
4.9.2	Count for the developed code	84
4.10	The Poisson equation for the pressure	87
5	Applications	89
5.1	Set up	89
5.2	Preliminary results	90
5.2.1	Two-dimensional lid-driven cavity flow	90
5.2.2	Dipole-wall interaction	98

CONTENTS

5.3	Three-dimensional cavity at $Re = 1000$ and $Re = 3200$. . .	99
5.4	Taylor-Green Vortex at $Re = 1600$	103
5.4.1	Scalability assessment	105
6	Conclusions and future works	115
6.1	CFD-related	115
6.1.1	The Poisson equation for the pressure	115
6.1.2	Spectral methods along periodic directions	117
6.1.3	On time integration	118
6.1.4	LES	119
6.2	Numerics-related	119
6.2.1	LU decomposition for tridiagonal matrices	119
6.2.2	SPIKE algorithm's reduced system	119
6.2.3	Coefficients of compact schemes	119
6.3	Programming-related	120
6.3.1	Serial and/or two dimensional run	120
6.3.2	Pointers to avoid temporary arrays	120
6.3.3	CDS by rows and JDS formats for banded matrices .	121
6.3.4	Condensation of MPI calls	122
6.3.5	Fortran-native vectorization and OpenMP-MPI . . .	123
6.3.6	Availability	123
A	Functions to compute finite differences' coefficients	125
	Bibliography	129

Chapter 1

Introduction

Turbulence is the ever-open challenge of fluid dynamics. Several questions regarding the most practical needs of aeronautical industry (control of transition points on airfoils, as well as the reduction of aircrafts' engine noise, just to mention two examples) rely upon the worldwide research on turbulence.

Numerical approaches to fluid dynamics in general, and turbulence in particular, are the most promising ones, since they can take advantage of both theoretical developments and advancements, and technological progress, in which parallel computing represents a major player. CFD solvers combine these two aspects.

1.1 The CFD Solver

The development from scratch of the three-dimensional, unsteady, incompressible, high-order Navier-Stokes parallel solver, which is the object of the thesis, was carried out with the objective of creating an original tool to perform CFD simulations, which is scalable, and maintainable and extendable, of course within a well-defined range of possibilities.

1.1.1 An in-lab code

Besides requiring an up-front investment plus an annual support/upgrade fee, commercial software used to perform simulations of fluid flow phenomena (e.g., FIDAP, FLUENT, CFX, STAR-CD, ...) are programmed to be can-do-everything tools — as the word *commercial* implies — where

it is possible to plug in new models or, more generally, new components to suit the specific physical problem under examination.

Sadly, these interventions frequently cause troubles in the basic code (e.g., stability- and convergence-related) which are not easy to solve, and can often be faced either by waiting for the next release of the software, or by devising a work-around, which may be a time consuming activity too. While the former approach does not guarantee that the software's developers will include the needed fixes and required features in the next release, the latter one hopefully leads to a solution which can speed-up the research activity; still this work-around is temporary solution, though, since it only fits that specific problem it was designed for, and inherits not at all the generality of the parent code, thus providing virtually no advantage in facing next troubles that are likely to be encountered. The reason for the weakness of these approaches is that we usually do not have the source code available, an handicap preventing us from attacking the intricacy at its roots, and forcing to shoe-horn the simulation of unique physical phenomena into the framework provided by the vendor. Without access to the details of a code feature, the purchaser encounters the real possibility that the computational tool is marginal for the required application. Conversely, in-lab codes are source codes, allowing unique physics sub-models to be accommodated more readily, and targeted to the specific needs of the user.

Another well-known issue, with simulation tools in general, is that different codes provide different answers to the same questions. Despite this being a problem common to both in-lab and commercial software, it has to be regarded with more suspicion in the latter case; indeed, the numerical solution of equations describing physical phenomena always carries characteristics of the selected numerical method into the results, especially when non-linear mechanisms are involved. That is to say, in either case no one can guarantee that there will be good solutions, but when dealing with an in-lab code, the author probably knows why he is not getting good answers and can address the problem either by freely and mindfully intervening in the source code, or by interpreting the differences, provided they are not too drastic, and "trimming" the conclusions accordingly.

Last but not least, when it comes to experimenting at a deeper level with the numerics underling the solver — e.g., changing or trimming the selected finite difference formulæ, casting a specific term of the equation in a new, undocumented special form, testing the impact of the order of

accuracy of boundary condition schemes on the global solution — the benefits springing from having an in-lab code are invaluable, since it naturally comes with the possibility of tweaking the tiniest details affecting the solution in critical points of the solver’s workflow, thus allowing the author (and the user as well, if a comprehensive user guide is provided) to isolate a bug during development stages, or a change when experimenting with existing features.

All this being said, having an in-lab code available does not replace timely access to a commercial code, but it does increase the options available to the analyst.

Concerning other open-source alternatives, four of the most known, used, and successful ones, are briefly discussed in the following.

Other open-source solvers

OpenFOAM [1–3], originally developed in 1989 as a commercial, closed-source product under the name of FOAM (Field Operation And Manipulation), was officially released as an open-source, object-oriented library for C++ in 2004, and has undergone continuous improvement since then. It is now a full-fledged, general purpose suite, freely available worldwide under the GNU Public License, with possible applications ranging from laminar incompressible flow to fully turbulent reacting compressible flow, or even to solving the Black-Scholes equation for the dynamics of financial market, thus becoming the de facto standard open-source tool for the solution of a variety of problems. It is shipped with a wide array of utilities for pre- and post-processing; with these free CFD tools, it is possible to conduct a full analysis from beginning to end using only the tools provided by the OpenFOAM distribution. Concerning the numerics, it uses finite volume and finite element formulations.

SU^2 (Stanford University Unstructured [4, 5]) suite, launched in January 2012, is an open-source collection of software tools, written in C++/Python, for solving partial differential equations and performing optimization problems. It lacks a complete suite of pre- and post-processing tools, but it is a better choice in specific compressible external flows in aeronautical problems [6].

NEK5000 [7, 8] is a solver featuring a state-of-the-art, scalable, high-order spectral element method, and has been developed for more than 30 years. Its applications span a wide range of fields, including fluid flow, thermal convection, combustion and magnetohydrodynamics. It is

written in old FORTRAN, and makes massive use of legacy features of the FORTRAN77 standard. A marginal comparison between that solver and the present one is drawn in Fig. 5.13.

Incompact3d [9, 10], made available around 2010 together with the 2DECOMP&FFT library [11], is a CFD solver with which the solver developed by the present author shares several characteristics: it solves the incompressible Navier-Stokes equations, discretizes the spatial operators through compact finite difference schemes, integrates the equation by means of a Runge-Kutta-based fractional step method in a parallelepipedal, structured computational domain. Two main differences are the use of *partially* staggered meshes, as opposed to the fully staggered meshes chosen by the present author, and the approach to parallelization, there accomplished through a two-dimensional, dynamic domain-decomposition, instead of the static three-dimensional decomposition adopted in the present work. A more extensive discussion is presented in Section 4.9.

The choice of the Modern Fortran programming language (see Section 1.1.2), the use of compact finite-difference schemes (cf. Chapter 2), and the adoption of a three-dimensional MPI domain-decomposition Sections 1.1.4 and 4.5, all together put this work in a position of originality with respect to the aforementioned alternatives, which, on the other hand, all benefit from a long-lasting development, as compared to the code presented here, which was developed on a one-person–three-years budget basis.

1.1.2 Fortran

Before digging in the reasons why the Fortran programming language was chosen, it is worth to recall that Fortran is a modern language, despite people and (to the author’s experience) computer scientists think of punch cards and code with line numbers, upon hearing “FORTRAN”. Indeed, although the original specification of the language was written in 1954 — and even then, it was an incredible leap forward from previous programming —, Fortran has undergone many revisions, the most known being relative to the 66, 77, 90, 95, 03, and 08 standards, each incorporating more and newer features as time passes (albeit slowly). This is to say, calling modern Fortran old is like calling C++ old because C was first developed around 1973.

In academic scientific community, Fortran remains a major tool and

will not be going away anytime soon. In a survey of Fortran users at the 2014 Supercomputing Convention [12], 100 % of respondents said they thought they would still be using Fortran in five years; in 2017 it was still the major language employed for high performance computing (HPC) [13].

Coming to the choice of the language, in the field of HPC — of which large scale numerical simulation is a subset — there are mainly two languages in use today, namely C++ and modern Fortran. The popular implementations of the OpenMP and MPI libraries for parallelizing code were developed for these two languages, so basically the choice was limited to these two options.

It is often said that the reason Fortran is still used is that it is fast. But is it the fastest? On most benchmarks, Fortran and C++ are the fastest, with the latter being generally more performant, except for those benchmarks involving predominant number-crunching over other aspects (e.g., n -body simulation, Fourier transforms, LU decomposition, . . .), which are very close to what is of interest in HPC as applied to CFD.

Indeed, Fortran is just natively suited for numerical programming. As a major example, the reference implementations of the low-level routines for linear algebra, collected in the BLAS and LAPACK libraries, which are part of the *de facto* omnipresent Intel’s Math Kernel Library (MKL), are coded in Fortran. Programs for numerical computing tend to have a large amount of numbers to crunch, which are typically arranged into arrays — this is especially true for CFD programs, as the one developed during this doctorate, since space-dependent physical quantities in a one-, two-, or three-dimensional space, are naturally thought of as stored in arrays of rank one, two, or three;¹ multidimensional arrays are first class citizens in Fortran, whose features were carefully designed to allow the compilers to recognize most spots for optimizations, and the programmer to access linear algebra tools through a transparent syntax. Indeed, it is often pretty straightforward to translate numerical kernels from MATLAB into Fortran, the former being the closest thing to a programming language the author had knowledge of at the time the Ph.D. started. The relative ease in recycling this previous knowledge was due to the Fortran language’s simplicity and the nice, succinct, and self-explanatory syntax of array operations, which MATLAB’s one is pretty much close to. Arrays can be copied, multiplied by a scalar, or multiplied together quite intuitively; almost all of the intrinsic functions in Fortran can take arrays as arguments, leading to

¹In Fortran, n -dimensional arrays are referred to as “arrays of rank n ”.

easy of use and very neat code; furthermore, dynamically allocating and deallocating arrays in Fortran is easy.

Contrary to the common misconception that Fortran uses 1-indexed arrays, this language in fact supports declaring arrays with lower indices that are zero or negative, an extremely useful feature, which the solver uses extensively in the declaration and use of space-dependent, processor-owned three-dimensional arrays, as explained in detail in Chapter 4.

Coming to the code, it is written in Fortran 2003 — plus a few features which are unique to the 2008 standard — and is parallelized through the use of MPI 3.1 standard’s advanced routines, as implemented by the OpenMPI library project. With this main priority being outlined, particular attention was paid to several key concepts: modularity and readability of the source code and, in turn, its re-usability and maintainability; ease of implementation of virtually any new explicit or implicit finite difference scheme; intuitive insertion of proper boundary condition; process-independent output; modern programming style and avoidance of deprecated old legacy Fortran constructs and features, so that the world wide web community is a reliable and active means to the quick solution of coding problems arising from the implementation of new modules in the code; last but not least, thorough comments, especially in critical sections of the code, explaining motives and possible expected weak links.

Lastly, it is mandatory to say that the performances of every language depend hugely on the used compiler, since a language is nothing more than a set of grammar rules and meaningful words. So, for a given code, the compiler is solely responsible to use the given language in order to translate the source code into a well-performing executable.

1.1.3 High order compact schemes

A main feature of the code, which is uncommon to the majority of the CFD solver, especially the commercial ones, is the use of the so-called *compact* high-order finite difference schemes for the discretization of the convective and diffusive operators in the Navier-Stokes equations.

A remarkable documentation exists concerning this class of implicit spatial schemes [14], which have been extensively used for flow problems. The rationale behind the choice of these schemes for the computation of interpolants and derivatives lies in their amenable properties, which offer a compromise between spectral schemes, which provide exact resolution

at all (representable) scales², but impose significant restrictions on the geometry, and finite difference schemes, which are highly flexible in the latter respect, at the price of poor resolution at high wavenumbers, which hardly improves as their formal accuracy is raised.

With these assets, compact schemes really match the needs of computational fluid dynamics, especially when applied to turbulence (whose signature is the wide range of spatial scales), and even more when the solver implements the large eddy simulation (LES), a context in which truncation errors cannot be allowed to overcome the already fragile contribution of the modelled terms.

One major drawback in using compact finite difference schemes lies in the fact that they are implicit, since every unknown on the mesh depends on all knowns, which translates into their application to be dependent on both the computation of a matrix-vector product, a step common to classical explicit finite differences, as well as the resolution of a banded linear system, which is inherently a serial procedure and, as such, cannot be parallelized straightaway as the former.

1.1.4 Parallel computing

The solver is meant to take advantage of parallel computing capabilities provided by modern hardware, whose exploitation is made available by modern features of programming languages and message-passing standards.

In line with this target, a static three-dimensional domain decomposition approach was chosen, in that it allows the maximum number of processes used in a three-dimensional simulation to increase linearly with the mesh refinement. As a consequence of this choice, no process holds all the data along any direction and, therefore, it cannot compute all the unknowns.

As raised earlier, when only explicit finite differences are concerned, this is in principle not a big deal, since the distributed computation of the matrix-vector product (and matrix-matrix as well, for that matter) is a standard problem of data parallelism, and many libraries are shipped with suitable routines. On the other hand, when it comes to compact schemes, the resolution of a banded linear system must be distributed among several processes. Past attempts can be grouped in three categories. One of this

²As well as the so-called spectral convergence.

is concerned with the use of asymmetric compact schemes at the boundary between processes, so that their systems of equations are decoupled and can be solved in parallel [15], at the price that the solution depends on the number of processes and the spectral-like properties of compact schemes are compromised. Another approach was the parallel implementation of the tridiagonal solver, such as the pipelined Thomas algorithm (PTA) where the idle time occurring during forward and backward substitutions is used to carry out non-local data-independent or local data-dependent computations [16], thus requiring a convoluted schedule of the communications and computations, which results in a trade-off between the efficiencies of the two. The third category comprises the so called pencil-transposition-based approaches, which are grounded in a non-static two-dimensional domain decomposition (of the three-dimensional domain), on the base of which each process performs all the required one-dimensional computations along one direction and then moves to the next direction [9], not before a very communication intensive transposition of the 2D decomposition is performed.

The problem of solving banded linear systems in a parallel framework is tackled by using the algorithm presented in detail in Chapter 3. This algorithm was developed by the author with no awareness that it already existed, since it had (and have had) no significant resonance in the fluid-dynamic community. Indeed, despite slight differences in the formalism adopted, the procedure coincides with the algorithm presented in 2006 and named SPIKE at that time [17], later reprised with slight variations and improvements [18–20], but fundamentally already existent a few years earlier and presented in different areas of research [21, 22], in one case as an improved version of the so called *parallel factorization*, originally developed in 1992, albeit in a different guise by Amodio et al. [23].

MPI

As disclosed in Section 1.1.4, MPI parallelization was accomplished by defining a static three-dimensional decomposition of the computational domain and explicit message passing. The adjective *static* is used to mean that this decomposition does not change during the run; in other words, the three-dimensional computational subdomain which is bounded to an MPI process, is also bounded to a fixed three-dimensional portion of the geometrical space.

Explicit data exchange at inter-process boundaries takes place twice for

each application of the interpolation/differentiation operators: the former exchange is aimed at making the appropriate known quantities available to each process, and depends on the bandwidth of the RHS compact scheme’s matrix; the latter is part of the SPIKE algorithm and does not depend on anything (invariably two data layers per process).

This is in contrast with the *dynamic* “pencil” domain decomposition — employed by Incompact3d [9] through the 2DECOMP&FFT library, developed from scratch by the same authors [11] and mildly used in the CFD community —, in which each process repeatedly undergoes, at each time step, a change of the geometrical space it handles, switching among the three possible configurations by a call to communication-intensive MPI subroutines. The implementation of this procedure is admittedly complex (at least more complex than the simpler, less performant, 1D domain decomposition [9]), even though the use of the routines is made very easy by a clear and neat API.

The topic is extensively treated in detail within Chapter 4.

1.2 Thesis layout

Chapter 2 explains the motives behind the choice of the compact schemes, comments the classical approach for the determination of their coefficients and for conducting a spectral analysis on them; then an algorithm is presented for the exact, algebraic determination of the coefficients of any finite difference scheme, being it compact or not, involving derivatives of any order located at any point, even contemplating the possibility of assigning an integral value to an interval between two points; finally, based on the experience acquired in the process of implementing that algorithm, compact schemes are meaningfully and, to the best of this author knowledge, originally commented, in a way that allows a reinterpretation of several breakthrough works presented in literature.

In Chapter 3 the SPIKE algorithm is derived as an extension of the basic idea of splitting the solution of a tridiagonal system in two. The truncated SPIKE algorithm is presented in Section 3.2, whereas another approach is briefly outlined in Section 3.3.

Chapter 4 is a dense walk-through of the code. It acknowledges the crucial importance of using a version control system, then describes the major components of the program, from the expected format of the input file, to an assessment of the predicted performances, by attempting a logical

travel through the underlying indexing of the variables, their interaction with the parallel framework, and the core datatypes used to store variables and operators. Possible weakest links are spotted and commented along the way, possibly referring to the last chapter for further details.

In Chapter 5 the CFD solver is tested against the three-dimensional Lid-Driven Cavity and Taylor-Green Vortex flows. The latter test is used to verify the parallel-independent results, as well as the parallel performances of the code. Preliminary tests, conducted by means of a MATLAB script, are presented beforehand.

Opportunities for the future improvement of the CFD solver in question are listed in Chapter 6.

Appendix A contains the Fortran and MATLAB specular functions for the computation of the coefficients of a general finite difference scheme. This routine has served to populate finite difference matrix operators.

Chapter 2

High accuracy schemes

Many physical phenomena possess a wide range of space and time scales, which must be taken into account when it comes to choose a numerical method, since the reproduction of motions governed by these wavelengths can be essential to the phenomenon, as is the case for turbulence, which is characterized by an even wider range of scales.

In the context of computational fluid dynamics, there are two diametrically opposed approaches for a fine enough representations of relevant motions, and they consist respectively in refining the mesh, and increasing the order of accuracy of the schemes;¹ both these attitudes have found supporters in the CFD community (e.g., [27–30] and [24–26, 31–34], respectively) since the former has the benefit of a low computational cost per mesh point but requires finer discretization, whereas the latter relaxes on the mesh by increasing the order of accuracy, at the costs of more floating point operations per point. In other words, whether a researcher should prefer the one or the other approach, has long been debated, since pros and cons are somewhat inverted in the two cases. Even so, among the low order methods, the mainstream second order schemes applied on a staggered mesh [30] have found solid ground in their ability to conserve kinetic energy, as well as mass and momentum, and for this reason they have been extensively used with remarkable success, even to LES [35, 36]. Nonetheless, the world research have crawled with papers about the use of high order schemes, and efforts were successfully made to extend the conservativeness of second-order approach to higher order schemes

¹The use of spectral method is avoided for its limitations [14, 24–26] (cf. Section 2.1.1).

[37]. Furthermore, there has always been an interest in the application of high-order schemes to LES [38–40], since the truncation error from a second-order discretization often overwhelms the contribution of the sub-grid model, thereby damaging LES at its foundations [41].

Another argument in favour of the application of higher order schemes on coarser meshes is that accepting the additional floating point computations implied by those schemes in exchange for a lower memory usage connected to a coarser grid is often a good deal, since the gain in speed from reducing memory access is often much greater than the cost of additional processing [42].

In the present thesis, this latter trend of relying on high-order schemes was preferred, and the specific selected schemes are the so-called *compact* schemes, as anticipated in Section 1.1.3.

2.1 Need for high resolution characteristics

In principle, high accuracy means high order of the truncation error, which can be obtained using high order classical finite difference approximation schemes. However, this path is not advisable, for several reasons outlined in Section 2.1.1, and the *compact* schemes are preferable, since they provide a better representation of the shorter length scales with respect to traditional finite difference schemes, still maintaining the same freedom in assigning geometry and boundary conditions, as the more common finite differences [14].

Notwithstanding, with the major exception of Incompact3D [9, 10], compact schemes have received relatively little resonance in the incompressible community, in contrast to the profitable use for the simulation of compressible flows in general [15, 33, 43], and aeroacoustics in particular [18].

2.1.1 Formal accuracy and high-frequency harmonics

Insisting on classical schemes, the so-called *formal* order of accuracy can be increased by including a greater number of mesh points in the stencil: for a 10th order approximation of the collocated first derivative on a uniform computational domain, an 11-point stencil is required. Such large stencils can produce undesired oscillation [44, 45], not to mention the problematic treatment of boundary conditions, which demands for asymmetric high

order formulæ.

After all, the order of accuracy is only informative of how, and how much, a refinement of the mesh reduces the truncation error afflicting the solution; however, increasing the number of grid points for large three-dimensional simulations is in general not a desirable option [33], since the memory limitations of the computer system are easily hit; it is desirable, on the contrary, to enhance the quality of the solution at a fair count of mesh points, and possibly regardless of the Fourier components of the given function. Increasing the formal order gives little benefit in this respect.

The requirement of high accuracy for a wider range of wavenumbers originally led to the development of spectral methods [46, 47], which have been used to carry out direct simulations of turbulent flows as well [48–50]; however, the paradigm underlying these methods does not sit easily with complex domains [14, 24–26]. It is worth to mention that, when a test-case provides for some periodic directions, those directions are most likely meshed uniformly, and physical properties does not vary along them; this is the perfect chance to use spectral methods alongside compact schemes; the code does not implement spectral methods along periodic directions, so far, but an upgrade is considered in Chapter 6.

As long as non-spectral methods are concerned, a more meaningful quantity to be taken into account than the formal order of accuracy when choosing a scheme, is the so-called *modified wavenumber* of a finite difference formula, whose meaning is anticipated in Section 2.1.2 and Fig. 2.1, and explained in Section 2.4. This characteristic is informative of how well the scheme can reproduce each representable length-scale of the solution. Under this point of view, traditional schemes come to be unsatisfactory, since they have poor spectral characteristics with respect to their compact counterparts, which offer a compromise between those inaccuracies and the unrivalled spectral schemes.

2.1.2 A first example: the Padé scheme

The most known fourth order scheme for the first derivative on a uniform mesh is, perhaps, the following one, which allows the explicit computational of a *single* unknown, based on *several* surrounding values of the function,

$$y'_j = \frac{2(y_{j+1} - y_{j-1})}{3h} - \frac{y_{j+2} - y_{j-2}}{12h}. \quad (2.1)$$

This formula is set on a five-point stencil and has a truncation error equal to $-h^4 f^{(5)}/90$. When applied to the simple case of a uniform periodic domain, Eq. (2.1) can be written for each mesh point and applied independently; in matrix notation, Eq. (2.1) can then be written as

$$\mathbf{y}' = \frac{1}{h} \mathbf{D} \mathbf{y}, \quad (2.2)$$

where \mathbf{D} is a cyclic Toeplitz banded differentiation matrix storing the coefficients $(+1/12, -2/3, 0, +2/3, -1/12)$ in its diagonals of indices from -2 to $+2$; the fact that \mathbf{y}' does not multiply any matrix — or, that it multiplies the identity matrix \mathbf{I} — expresses the mutual *independence* of its components.

The corresponding same-order compact scheme is known the by the name of Padé scheme, which reads

$$\frac{1}{4} y'_{j-1} + y'_j + \frac{1}{4} y'_{j+1} = \frac{3}{4h} (y_{j+1} - y_{j-1}), \quad (2.3)$$

where the narrower three-point stencil with coordinates $(-h, 0, +h)$ accommodates both three values of the derivatives at $x = -h, 0, h$, and two values of the function at $x = \pm h$, which are most likely unknown and known quantities respectively. Concerning the truncation error, in order to allow a fair comparison with Eq. (2.1), it must determined once Eq. (2.3) is cast in a way that all the coefficients at the LHS sum up to 1 [51]. In this case, the error is $-h^4 f^{(5)}/180$, that is half Eq. (2.1)'s error. Since more unknown values are involved in the LHS of Eq. (2.3), none of them can be computed independently, thus making the relation implicit; to solve Eq. (2.3), other equations involving those quantities must be added, in order to form a determined system of equations for the unknowns. For instance, with reference to the same simple case of a uniform periodic domain as before, Eq. (2.3) can be written for each point, thus resulting in a cyclic tridiagonal system,

$$\mathbf{A} \mathbf{y}' = \frac{1}{h} \mathbf{B} \mathbf{y}, \quad (2.4)$$

with obvious meaning of the symbols.² Here, the differentiation matrix

²It is worth to mention that Eqs. (2.2) and (2.4) are general formulæ that hold for the non-periodic and/or non-uniform case as well; what changes is the actual content of the matrices: boundary conditions are encoded in an appropriate number of top and bottom rows, which can increase the bandwidths, and the Toeplitzness is kept only in the inner rows, as long as the mesh is kept uniform, otherwise it is lost.

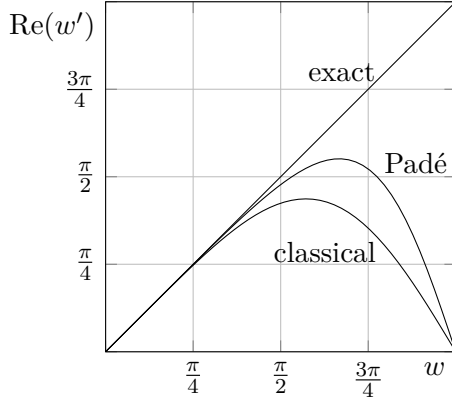


Figure 2.1: Modified wavenumber w' relative to Eqs. (2.1) and (2.3) as compared to the actual wavenumber w , which is real for symmetric formulæ. In the case of the Padé scheme, the modified wavenumber sticks to the bisector for a wider range of wavenumbers, as compared to the classical scheme.

corresponding to \mathbf{D} is $\mathbf{A}^{-1}\mathbf{B}$, which is full, as a consequence of the fact that the unknowns are all coupled together.³

A major difference between Eqs. (2.1) and (2.3) lies in the their modified wavenumber w' (a thorough investigation of which is conducted later, in Section 2.4), depicted in Fig. 2.1 as function of the actual wavenumber w , since the one relative to the compact scheme stays close to w over a longer interval than that relative to the classical fourth order scheme.

2.2 A schematic graphical representation

In Fig. 2.2, an original representation of the stencils relative to Eqs. (2.1) and (2.3) is sketched. From now on, the stencils are represented in a similar manner, using the following notation:

- an horizontal line is drawn to represent the 1-D mesh;
- for each mesh point where a quantity is prescribed (resp. requested), a vertical stem is drawn above (resp. below) the horizontal line, at its coordinate relative to a reference uniformly-spaced mesh;

³This newly introduced terse matrix notation proves to be far more convenient then writing single equations in full, especially when the specific schemes are not relevant to the discussion, e.g., Chapter 3.

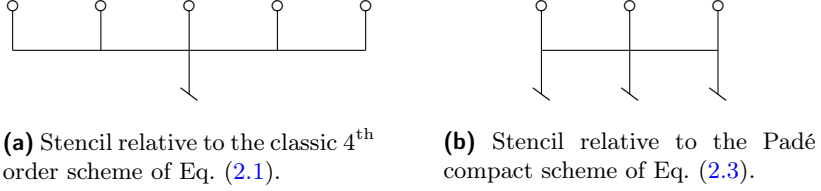


Figure 2.2: Stencils relative to Eqs. (2.1) and (2.3), sketched accordingly to the convention proposed by Coppola [52] and later reprised by De Angelis [53].

- the stem is drawn even if the corresponding coefficient is zero, since the latter condition is, in general, an episodic outcome of the specific metric of the mesh (e.g., the symmetry);
- symbols at the end of the stems are drawn to indicate what quantity is located there: a single circle (\circ) for the function, and as many oblique dashes (\diagdown) as the order of differentiation for the derivative;
- each coefficient is intended to have the value that maximizes the order of accuracy, unless it is explicitly typeset close to the symbol.

These sketches were first introduced by Coppola [52] and recently proposed again with slight variations [51]. One advantage of such a notation is that it immediately conveys several aspects relative to the represented scheme, among which

- the equation of the corresponding scheme,
- the width of both left and right hand side’s stencils,
- the closest-to-boundary point where the scheme can be used without protruding beyond the boundary,
- the symmetry,
- the order of accuracy.

2.3 General formulation for the symmetric case

In this section the assumption of uniform mesh is done — without which symmetry cannot occur —, and a general strategy for obtaining symmetric finite difference formulæ is derived, which includes the schemes of

Section 2.1.2. Firstly, some useful operators are defined in Section 2.3.1; secondly, explicit symmetric schemes, within which Eq. (2.1) can be included, are derived in Section 2.3.2; thirdly, the procedure is “upgraded” in Section 2.3.3 in order to obtain compact schemes that generalize Eq. (2.3).

2.3.1 Convenient linear operators

In this respect, it proves convenient (cf. [52]) to define several linear operators acting on the numeric array $\mathbf{y} = (y_j)_j$; first the I (identity) and E (shift) elementary operators are defined,

$$Iy_j \doteq y_j \quad Ey_j \doteq y_{j+1};$$

then, by sums and differences,

$$\begin{aligned} \delta_k &\doteq E^k - E^{-k}, & k \geq 1 & \quad (\text{central difference}) \\ \mu_k &\doteq E^k + E^{-k}, & k \geq 0 & \quad (\text{central sum}) \\ \Delta_k &\doteq E^k - I, & k \geq 1 & \quad (\text{forward difference}) \\ \nabla_k &\doteq I - E^{-k}, & k \geq 1 & \quad (\text{backward difference}). \end{aligned}$$

These operators can be applied to y_j once to give

$$\begin{aligned} \delta_k(y_j) &= y_{j+k} - y_{j-k} \\ \mu_k(y_j) &= y_{j+k} + y_{j-k} \\ \Delta_k(y_j) &= y_{j+k} - y_j \\ \nabla_k(y_j) &= y_j - y_{j-k}, \end{aligned}$$

twice to give

$$\begin{aligned} \delta_k^2(y_j) &= \delta_k(\delta_k(y_j)) = y_{j+k} - 2y_j + y_{j-k} \\ \mu_k^2(y_j) &= \mu_k(\mu_k(y_j)) = y_{j+k} + 2y_j + y_{j-k} \\ \Delta_k^2(y_j) &= \Delta_k(\Delta_k(y_j)) = y_{j+2k} - 2y_{j+k} + y_j \\ \nabla_k^2(y_j) &= \nabla_k(\nabla_k(y_j)) = y_j - 2y_{j-k} + y_{j-2k}, \end{aligned}$$

and so on.

2.3.2 Explicit symmetric formulæ

The Taylor series expansion of $y_{j\pm k} = f(x \pm h)$ is the following

$$y_{j\pm k} = y_j \pm (kh)y'_j + \frac{(kh)^2}{2!}y''_j \pm \frac{(kh)^3}{3!}y'''_j + \dots;$$

subtracting the backward expansion (−) from the forward (+), recalling the operators defined in Section 2.3.1, and dividing by $2kh$, one obtains the relation

$$\frac{\delta_k(y_j)}{2kh} = y'_j + \frac{(kh)^2}{3!} y_j''' + \frac{(kh)^4}{5!} y_j^{(5)} + \frac{(kh)^6}{7!} y_j^{(7)} + \dots, \quad \text{for } k \geq 1, \quad (2.5)$$

which is the starting point for building the explicit formulæ for the first derivative y'_j , since linear combinations of Eq. (2.5) over k can be written,

$$\sum_{k=1}^n \alpha_k \frac{\delta_k(y_j)}{2kh} = \sum_{k=1}^n \alpha_k \left[y'_j + \frac{(kh)^2}{3!} y_j''' + \frac{(kh)^4}{5!} y_j^{(5)} + \frac{(kh)^6}{7!} y_j^{(7)} + \dots \right], \quad (2.6)$$

and subtracted from the sought finite difference formula,

$$\sum_{k=1}^n \alpha_k \frac{\delta_k(y_j)}{2kh} = y'_j,$$

and the coefficients α_k be determine by requiring that the successive terms of the resulting truncation error cancel out,

$$\begin{aligned} \left[\sum_{k=1}^n \alpha_k - 1 \right] y'_j h^0 + \left[\sum_{k=1}^n \alpha_k \frac{k^2}{3!} \right] y_j''' h^2 + \\ + \left[\sum_{k=1}^n \alpha_k \frac{k^4}{5!} \right] y_j^{(5)} h^4 + \left[\sum_{k=1}^n \alpha_k \frac{k^6}{7!} \right] y_j^{(7)} h^6 + \dots \quad (2.7) \end{aligned}$$

In the trivial case that $n = 1$, all summations collapse to one term, only the first term of Eq. (2.7) can be set to zero, and the well known second order 2-point central formula is recovered

$$y'_j = \frac{y_{j+1} - y_{j-1}}{2h}. \quad (2.8)$$

For $n = 2$, the second addend in the square brackets is allowed to be canceled by choosing $\alpha_1 = \frac{4}{3}$ and $\alpha_2 = -\frac{1}{3}$, thus obtaining exactly Eq. (2.1). Linear combinations with higher values of n lead to higher order schemes, whose weighs were collected by Fornberg [54, Table 1].

2.3.3 Compact symmetric formulæ

Eq. (2.3) differs from Eq. (2.1) for having more terms in the LHS, which suggests to write the Taylor expansion of y'_j ,

$$y'_{j\pm k} = y'_j \pm (kh)y''_j + \frac{(kh)^2}{2!}y'''_j \pm \frac{(kh)^3}{3!}y^{(4)}_j + \dots;$$

and then to apply the central sum operator μ_k , thus obtaining

$$\frac{\mu_k(y'_j)}{2} = y'_j + \frac{(kh)^2}{2!}y'''_j + \frac{(kh)^4}{4!}y^{(5)}_j + \frac{(kh)^6}{6!}y^{(7)}_j + \dots, \quad \text{for } k \geq 0. \quad (2.9)$$

Here the same addends of Eq. (2.5) appear — differing only by the denominators' factorials —, so one can argue that a linear combination of Eq. (2.9) over k ,

$$\sum_{k=0}^m \alpha'_k \frac{\mu_k(y'_j)}{2} = \sum_{k=0}^m \alpha'_k \left[y'_j + \frac{(kh)^2}{2!}y'''_j + \frac{(kh)^4}{4!}y^{(5)}_j + \frac{(kh)^6}{6!}y^{(7)}_j + \dots \right], \quad (2.10)$$

can contribute in canceling out other terms of the truncation error. Indeed, the LHSs of Eqs. (2.6) and (2.10) can be required to be equal to each other, thus obtaining⁴

$$y'_j + \sum_{k=1}^m \alpha'_k \frac{\mu_k(y'_j)}{2} = \sum_{k=1}^n \alpha_k \frac{\delta_k(y_j)}{2kh}, \quad (2.11)$$

which is the sought symmetric compact finite difference formula for the first derivative, whose truncation error is clearly the difference between the RHS of Eqs. (2.6) and (2.10),

$$\begin{aligned} y'_j + \sum_{k=1}^m \alpha'_k \left[y'_j + \frac{(kh)^2}{2!}y'''_j + \frac{(kh)^4}{4!}y^{(5)}_j + \frac{(kh)^6}{6!}y^{(7)}_j + \dots \right] + \\ - \sum_{k=1}^n \alpha_k \left[y'_j + \frac{(kh)^2}{3!}y'''_j + \frac{(kh)^4}{5!}y^{(5)}_j + \frac{(kh)^6}{7!}y^{(7)}_j + \dots \right]. \end{aligned}$$

⁴The assumption $\alpha'_1 \doteq 1$ is made, so as to avoid the indeterminacy on the coefficients, and the corresponding term y'_j is taken out of the $\sum_{k=0}^m$ summation.

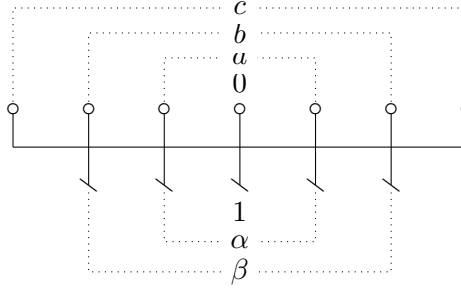


Figure 2.3: Stencil of Eq. (2.11) for $m = 2$ and $n = 3$. The coefficients are renamed in agreement with Lele [14], and allow for a 10th order formal accuracy.

This error can be more conveniently refactored by collecting coefficients of common degree with respect to h :

$$\begin{aligned}
 & \left[1 + \sum_{k=1}^m \alpha'_k - \sum_{k=1}^n \alpha_k \right] y'_j h^0 + \left[\sum_{k=1}^m \alpha'_k \frac{k^2}{2!} - \sum_{k=1}^n \alpha_k \frac{k^2}{3!} \right] y'''_j h^2 + \\
 & + \left[\sum_{k=1}^m \alpha'_k \frac{k^4}{4!} - \sum_{k=1}^n \alpha_k \frac{k^4}{5!} \right] y_j^{(5)} h^4 + \\
 & + \left[\sum_{k=1}^m \alpha'_k \frac{k^6}{6!} - \sum_{k=1}^n \alpha_k \frac{k^6}{7!} \right] y_j^{(7)} h^6 + \dots \quad (2.12)
 \end{aligned}$$

The coefficients α_k and α'_k to be plugged into Eq. (2.11) are determined in order to cancel out successive bracketed groups of Eq. (2.12) — thus giving a scheme of maximum order $2(m+n)$ —, or to fulfill other requirements while relaxing on formal accuracy. This formulation for first derivative is the most general for the symmetric case.

When $m = n = 1$ is chosen, one can set to zero at most the first two bracketed terms of Eq. (2.12), thus enforcing the fourth order accuracy and obtaining again the Padé scheme of Eq. (2.3), whereas when the choice is $m = 2$ and $n = 3$, the schemes presented by Lele [14, Sec. 2.1] are obtained, whose stencil is depicted in Fig. 2.3; an explicit scheme of the same order of accuracy would have the stencil depicted in Fig. 2.4.

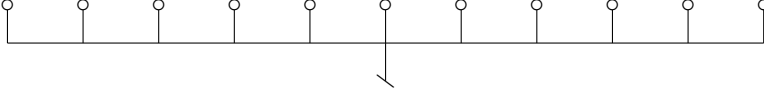


Figure 2.4: Stencil of the 10th order explicit scheme.

2.4 Spectral analysis

This section concerns the Fourier (or spectral) analysis of the errors associated with both explicit and compact finite difference schemes so far presented, in order to assess their resolution characteristics, i.e., their accuracy in representing the result over the full range of length scales realizable on a given mesh — this notion of resolution is akin to, but more general than, that of *intervals per wavelength* presented by Swartz et al. [55]. Fourier analysis consists in applying the numerical scheme to a periodic dependent variable and comparing the outcome with the known exact result under the specific operation.

Let the domain be $[0, L]$, over which the dependent variable $f(x)$ is periodic, i.e., $f(0) = f(L)$; let it be divided into N parts by $N + 1$ points x_j uniformly spaced by a step $h = L/N$. Since $f(x)$ is periodic and real valued, it can be decomposed as the sum of complex modes of wavenumber k , multiplied by Fourier coefficients $\hat{f}_k \in \mathbb{C}$ (which result to be complex conjugates with respect to k , i.e., $\hat{f}_k = \hat{f}_{-k}^*$ for $k = 1, 2, \dots, N/2$ and $\hat{f}_0 = \hat{f}_0^* \in \mathbb{R}$),

$$f(x) = \sum_{k=-N/2}^{+N/2} \hat{f}_k e^{i \frac{2\pi h k}{L} \frac{x}{h}}, \quad (2.13)$$

The maximum wavenumber $|k| = N/2$ corresponds to the highest frequency component allowed on the mesh, i.e., the harmonic function with period $2h$, as depicted in Fig. 2.5.⁵ By means of Euler's formula, Eq. (2.13) can be also rewritten as

$$f(x) = \hat{f}_0 + 2 \sum_{k=1}^{+N/2} \operatorname{Re}(\hat{f}_k) \cos\left(\frac{2\pi h k}{L} \frac{x}{h}\right) - 2 \sum_{k=1}^{+N/2} \operatorname{Im}(\hat{f}_k) \sin\left(\frac{2\pi h k}{L} \frac{x}{h}\right).$$

For clarity, the general k^{th} mode can be cast in a simpler form by defining a non-dimensional abscissa $\tilde{x} = \frac{x}{h}$ and a scaled wavenumber $w_k = \frac{2\pi h k}{L}$ —

⁵To take into account odd N numbers — then the sinusoid of Fig. 2.5 cannot be reproduced — the summation bounds could be written more precisely as $\pm \lfloor N/2 \rfloor$.

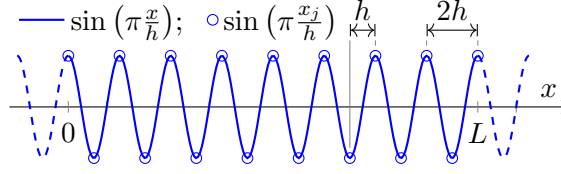


Figure 2.5: Highest-frequency sinusoid allowed on a mesh with an even number of points (the point at $x = L$ coincides with the one in $x = 0$).

the adjective is often dropped hereinafter, as long as the subscript k — as follows,

$$\widehat{f}_k e^{i \frac{2\pi h k}{L} \frac{x}{h}} = \widehat{f}_k e^{i w_k \tilde{x}}.$$

The exact first derivative with respect to x of the mode of reduced wavenumber w is

$$i \frac{w}{h} \widehat{f}_k e^{i w \tilde{x}},$$

whereas the numeric differentiation *mimics* it, thus giving

$$i \frac{w'}{h} \widehat{f}_k e^{i w \tilde{x}},$$

where w' is the *modified* scaled wavenumber — the definition of k' being obvious — whose closed expression (if exists) is determined by the numerical scheme. Clearly, the more w' resembles w throughout its domain $[0, \pi]$, the more the specific mode is well resolved, so it is of interest to plot w' against w , to visualize the spectral *inaccuracy* as the “gap” between the actual curve $w(w')$ and the ideal one, $w = w'$. This “gap” can be quantified as the error e [14]

$$e(w) = \frac{|w'(w) - w|}{w},$$

and bounded by a given tolerance ε , so that modes of wavenumber w for which $e(w) < \varepsilon$ are considered well resolved, whereas those for which $e(w) > \varepsilon$ are considered poorly resolved. Since a constant function (which corresponds to $w = 0$) is exactly resolved by any consistent numerical scheme, an expected general behavior is that low-wavenumber sinusoids are well resolved.

The physical implications of the *altered* wavenumber — using an adjective far more meaningful than *modified* — can be shown by making a

single mode evolve through the simple propagation equation

$$\frac{\partial f}{\partial t} + \frac{\partial f}{\partial x} = 0, \quad \text{with } t \in \mathbb{R}^+ \text{ and } x \in \mathbb{R},$$

whose exact solution is any function of $\xi = x - t$. In this sense, the single spatial mode (cf. [56]) $f(x, 0) = \widehat{f}(0)e^{i\frac{w}{h}x}$ is readily proved to evolve in time like

$$f(x, t) = \widehat{f}(0)e^{i\frac{w}{h}(x-t)}. \quad (2.14)$$

This is the initial ($t = 0$) sinusoid propagating at a phase speed $v_p = 1$. On the other hand, the numerical solution is the following,

$$f(x, t) = \widehat{f}(0)e^{i\frac{w}{h}(x-\frac{w'}{w}t)},$$

that is, the same initial wave propagating at a different speed. This speed can be complex, as is the case for asymmetric formulæ, and so the wavenumber can be written as $w' = w'_r + iw'_i$, thus finding that

$$f(x, t) = \widehat{f}(0)e^{\frac{w'_i}{h}t}e^{i\frac{w}{h}(x-\frac{w'_r}{w}t)}. \quad (2.15)$$

Comparing Eq. (2.15) to Eq. (2.14) makes it clear that the discrepancy between w'_r and w is indicative of a wrong phase speed ($v'_p = \frac{w'_r}{w} \neq v_p$), whereas the fact that $w'_i \neq 0$ causes the amplitude to change in time. In CFD community, these two effects are widespread and meaningfully referred to as *dispersion* and *dissipation*. In particular, note that the latter is potentially catastrophic, since it implies, in the case that $w'_i > 0$, an unbounded growth of the amplitude of the mode.

2.4.1 Application to compact schemes

Eq. (2.11) can be particularized in the case that $m = 2$ and $n = 3$, and redefine coefficients properly ($\alpha'_1 = \alpha$, $\alpha'_2 = \beta$, $a_1 = a$, $a_2 = b$ and $a_3 = c$), thus obtaining exactly the same scheme of Lele [14]:

$$\begin{aligned} \beta f'_{j-2} + \alpha f'_{j-1} + f'_j + \alpha f'_{j+1} + \beta f'_{j+2} = \\ a \frac{f_{j+1} - f_{j-1}}{2h} + b \frac{f_{j+2} - f_{j-2}}{4h} + c \frac{f_{j+3} - f_{j-3}}{6h}. \end{aligned} \quad (2.16)$$

By inserting a single mode $f_j = \widehat{f}_k e^{iw\tilde{x}_j}$ in Eq. (2.16), using Euler's formula, and recalling that $f_{j+k} = f(x_{j+k}) = f(x_j + kh) = f(x_j)e^{iwh} = f_j e^{iwh}$

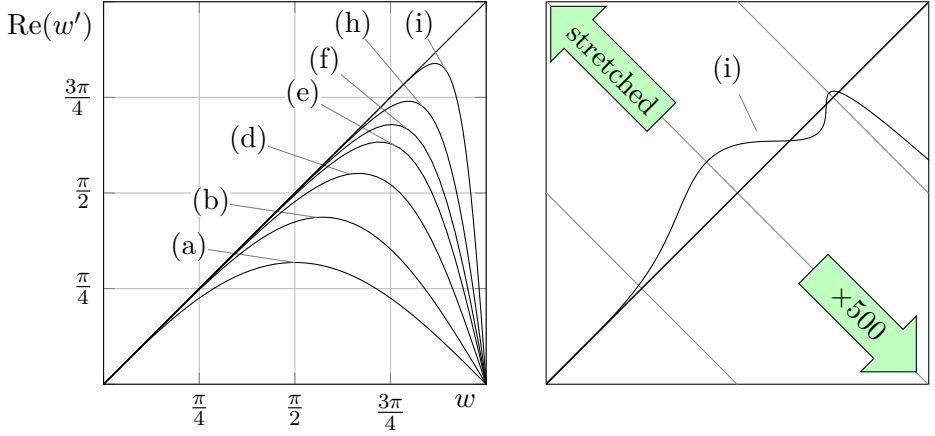


Figure 2.6: *Left:* Plot of the real part of the modified wavenumber, w'_r , vs. wavenumber w for first derivative approximations: (a) 2nd order explicit; (b) 4th order explicit; (d) 4th order tridiagonal (Padé); (e) 6th order tridiagonal; (f) 8th order tridiagonal; (h) 10th order pentadiagonal.

Right: w - w' plane stretched orthogonally to the bisector by a factor 500; each grey line results from the overlapping — to eye precision — of one horizontal and one vertical grid line on the *left*, due to the heavy distortion.

(and similarly for f'_{j+k}), the expression of the modified wavenumber, for all the schemes embodied by Eq. (2.16), is obtained

$$w'(w) = \frac{a \sin(w) + \frac{b}{2} \sin(2w) + \frac{c}{3} \sin(3w)}{1 + 2\alpha \cos(w) + 2\beta \cos(2w)}. \quad (2.17)$$

For several schemes, spectral curves are traced in the *left* plot of Fig. 2.6. The advantage in using compact schemes over classical explicit finite difference is clear and, already for the mere Padé scheme (d), it is remarkable, not to mention the spectral-like pentadiagonal one (i).

All the curves but the last, (i), are obtained, as already said, imposing the truncation error to be of the highest order possible. Scheme (i), on the other hand, is obtained giving up on formal accuracy — which is just 4th order, like the Padé scheme (d) — and calibrating coefficients of Eq. (2.17) so as to extend the good agreement with the bisector towards higher values of w . The *Right* plot of Fig. 2.6 shows, for one of the schemes depicted, that the curve $w'(w)$ can repeatedly intersect the bisector.

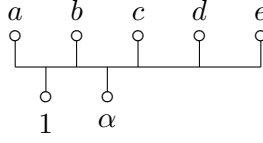


Figure 2.7: Stencil of a symmetric 6th order interpolation scheme.

2.5 Asymmetric compact schemes

This section deals with the computation of the coefficients relative to asymmetric compact schemes, whose use is unavoidable, as soon as boundary conditions are concerned.

The coefficients relative to typical asymmetric schemes for the computation of interpolant, first derivative, and second derivative are determined, in Sections 2.5.1, 2.5.2 and 2.5.3 respectively, and some hints are given which form the basis for the algorithm developed in Section 2.6, which is aimed at the computation of those coefficients in the most general case of compact finite differences. As anticipated in Section 1.2, this algorithm is implemented in the CFD code, where it is used to populate the variables containing the finite difference operators.

The Fortran implementation of the algorithm is presented in Listing A.1, together with its more succinct MATLAB version in Listing A.2.

2.5.1 Interpolation schemes on staggered stencils

Interpolation makes sense when the stencils on left and right hand sides are staggered to each other, so a plausible stencil for a 6th order boundary scheme of this kind is the one depicted in Fig. 2.7, where the staggering is supposed to be $h/2$. Such a scheme can be used, for instance, to interpolate a function for its first two cell-centered values, based on those at the first five faces, starting from the left boundary. The corresponding equation is the following

$$f_{\frac{1}{2}} + \alpha f_{\frac{3}{2}} = af_0 + bf_1 + cf_2 + df_3 + ef_4. \quad (2.18)$$

Upon expanding the terms in Taylor series about the point x_0 ,

$$\begin{aligned}
& \left[f_0 + \frac{h}{2} f'_0 + \left(\frac{h}{2} \right)^2 \frac{f''_0}{2!} + \left(\frac{h}{2} \right)^3 \frac{f'''_0}{3!} + \left(\frac{h}{2} \right)^4 \frac{f^{(4)}_0}{4!} + \left(\frac{h}{2} \right)^5 \frac{f^{(5)}_0}{5!} + \dots \right] + \\
& + \alpha \left[f_0 + \frac{3h}{2} f'_0 + \left(\frac{3h}{2} \right)^2 \frac{f''_0}{2!} + \left(\frac{3h}{2} \right)^3 \frac{f'''_0}{3!} + \left(\frac{3h}{2} \right)^4 \frac{f^{(4)}_0}{4!} + \left(\frac{3h}{2} \right)^5 \frac{f^{(5)}_0}{5!} + \dots \right] = \\
& = a f_0 + b \left[f_0 + h f'_0 + h^2 \frac{f''_0}{2!} + h^3 \frac{f'''_0}{3!} + h^4 \frac{f^{(4)}_0}{4!} + h^5 \frac{f^{(5)}_0}{5!} + \dots \right] + \\
& + c \left[f_0 + 2h f'_0 + (2h)^2 \frac{f''_0}{2!} + (2h)^3 \frac{f'''_0}{3!} + (2h)^4 \frac{f^{(4)}_0}{4!} + (2h)^5 \frac{f^{(5)}_0}{5!} + \dots \right] \\
& + d \left[f_0 + 3h f'_0 + (3h)^2 \frac{f''_0}{2!} + (3h)^3 \frac{f'''_0}{3!} + (3h)^4 \frac{f^{(4)}_0}{4!} + (3h)^5 \frac{f^{(5)}_0}{5!} + \dots \right] \\
& + e \left[f_0 + 4h f'_0 + (4h)^2 \frac{f''_0}{2!} + (4h)^3 \frac{f'''_0}{3!} + (4h)^4 \frac{f^{(4)}_0}{4!} + (4h)^5 \frac{f^{(5)}_0}{5!} + \dots \right],
\end{aligned}$$

the addends of common degree in h can be collected, and their coefficients set to zero. Clearly, since the 6 unknowns (α, a, b, c, d, e) are to be determined, only 6 conditions can be set, thus obtaining a 6×6 system of equations, enforcing 6th order accuracy. It is straightforward to verify that the k^{th} equation, for $0 \leq k \leq 5$, is the following, obtained by setting to zero the coefficient multiplying $h^k f_0^{(k)}$,

$$\left[\left(\frac{1}{2} \right)^k + \left(\frac{3}{2} \right)^k \alpha \right] = \left[0^k a + 1^k b + 2^k c + 3^k d + 4^k e \right], \quad (2.19)$$

where the $1/k!$ terms have been simplified since they appear both in the left and right hand sides, because of the same order of differentiation of all the terms involved in Eq. (2.18). It is worth to observe now that for each monomial in Eq. (2.18), there is one in Eq. (2.19) which is the product of the corresponding unknown coefficient by the corresponding x coordinate, the latter to the power k .

With regards to the solution by hand of Eq. (2.19), multiplying both sides by 2^k , a simpler form is obtained,

$$-3^k \alpha + 0^k a + 2^k b + 4^k c + 6^k d + 8^k e = 1,$$

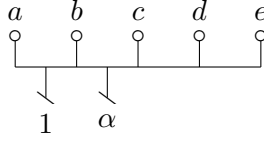


Figure 2.8: Stencil of a asymmetric 5th order first differentiation scheme on a staggered stencil.

which is easily expanded, with respect to k , in a matrix frame

$$\begin{pmatrix} -1 & 1 & 1 & 1 & 1 & 1 \\ -3 & 0 & 2 & 4 & 6 & 8 \\ -9 & 0 & 4 & 16 & 36 & 64 \\ -27 & 0 & 8 & 64 & 216 & 512 \\ -81 & 0 & 16 & 256 & 1296 & 4096 \\ -243 & 0 & 32 & 1024 & 7776 & 32768 \end{pmatrix} \times \begin{pmatrix} \alpha \\ a \\ b \\ c \\ d \\ e \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

The solution to this system of equations, can be found using MATLAB's symbolic tool,

$$\begin{pmatrix} \alpha \\ a \\ b \\ c \\ d \\ e \end{pmatrix} = \begin{pmatrix} 7/3 \\ 35/192 \\ 35/16 \\ 35/32 \\ -7/48 \\ 1/64 \end{pmatrix}.$$

2.5.2 First differentiation schemes on staggered stencils

Such a scheme is useful when the Harlow-Welch [30] arrangement of variables is employed. One possible stencil for such a scheme looks very similar to the preceding one, except that derivatives (\setminus) replace interpolations (\circ) in the LHS (lower part), but they are different in respect of the order of accuracy as well, since the scheme just presented is 5th order accurate, because of the factor $1/6$. The equation corresponding to Fig. 2.8 is the following,

$$f'_{\frac{1}{2}} + \alpha f'_{\frac{3}{2}} = \frac{1}{h}(af_0 + bf_1 + cf_2 + df_3 + ef_4), \quad (2.20)$$

each term of which can be expanded in Taylor series about x_0 , and the resulting terms can be collected in the same fashion as done in Section 2.5.1,

thus obtaining

$$\frac{1}{(k-1)!} \left[\left(\frac{1}{2} \right)^{k-1} + \left(\frac{3}{2} \right)^{k-1} \alpha \right] = \frac{1}{k!} [0^k a + 1^k b + 2^k c + 3^k d + 4^k e],$$

for $0 \leq k \leq 5$. (2.21)

where, for convenience and brevity, given any integer $n < 0$, the number 0 is denoted by the symbol $1/n!$. It can be noted now that the RHS of Eq. (2.21) is exactly the same as that of Eq. (2.19) — before $1/k!$ is simplified there —, just like the RHS of Eqs. (2.18) and (2.20). For what concerns the LHS of the two equations, they are akin as well, except that $k-1$ takes the place of k (both as exponent and as factorial), since it comes from first derivative terms.

As regards the solution by hand, multiply Eq. (2.21) by $3 \cdot 2^k k!$ yields

$$2k(1 + 3^{k-1}\alpha) = 0^k a + 2^k b + 4^k c + 6^k d + 8^k e,$$

which expands to the following matrix form

$$\begin{pmatrix} 0 & 3 & 3 & 3 & 3 & 3 \\ -6 & 0 & 6 & 12 & 18 & 24 \\ -36 & 0 & 12 & 48 & 108 & 192 \\ -162 & 0 & 24 & 192 & 648 & 1536 \\ -648 & 0 & 48 & 768 & 3888 & 12288 \\ -2430 & 0 & 96 & 3072 & 23328 & 98304 \end{pmatrix} \times \begin{pmatrix} \alpha \\ a \\ b \\ c \\ d \\ e \end{pmatrix} = \begin{pmatrix} 0 \\ 6 \\ 12 \\ 18 \\ 24 \\ 30 \end{pmatrix},$$

whose solution is found to be,

$$\begin{pmatrix} \alpha \\ a \\ b \\ c \\ d \\ e \end{pmatrix} = \begin{pmatrix} 71/9 \\ -127/216 \\ -49/6 \\ 37/4 \\ -29/54 \\ 1/24 \end{pmatrix}.$$

2.5.3 Second differentiation schemes on collocated stencils

Such a scheme, just like the preceding, is useful when the Harlow-Welch arrangement of variables is employed, since the second derivatives of the

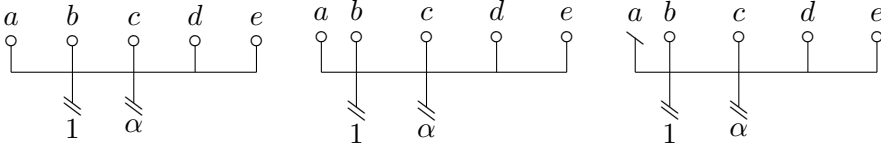


Figure 2.9: Three possible stencils of a asymmetric 4th order second differentiation scheme. *Left:* first node not staggered, Dirichlet. *Middle:* first node $h/2$ -staggered, Dirichlet. *Right:* first node $h/2$ -staggered, Neumann.

components of the velocity (coming from the Laplace operator) are required in the same location of the components themselves. Three plausible stencils are depicted in Fig. 2.9: the *left* one is used when a Dirichlet boundary condition for the concerned variable is prescribed on a point where the variable itself is located; if this BC is imposed on a $h/2$ -staggered node, then the *middle* stencil is used; when a Neumann condition is set in this staggered node, then the stencil on the *right* is chosen.

The system of equation for the coefficients of the scheme corresponding to the stencil on the *right* is derived as follows. The corresponding finite difference equation is

$$f''_{\frac{1}{2}} + \alpha f''_{\frac{3}{2}} = \frac{1}{h^2} \left(ahf'_0 + bf_{\frac{1}{2}} + cf_{\frac{3}{2}} + df_{\frac{5}{2}} + ef_{\frac{7}{2}} \right),$$

or, by a convenient shift of all indices by $1/2$,

$$f''_0 + \alpha f''_1 = \frac{1}{h} af'_{-\frac{1}{2}} + \frac{1}{h^2} (bf_0 + cf_1 + df_2 + ef_3). \quad (2.22)$$

It is straightforward to derive the following modified equation,

$$\begin{aligned} \frac{1}{(k-2)!} [0^{k-2} + 1^{k-2}\alpha] &= \frac{1}{(k-1)!} \left[\left(-\frac{1}{2} \right)^{k-1} a \right] + \\ &+ \frac{1}{k!} [0^k b + 1^k c + 2^k d + 3^k e], \quad \text{for } 0 \leq k \leq 5, \end{aligned} \quad (2.23)$$

in which factorials and powers are clearly equal to k minus the order of differentiation they are related to. The system of equation is easily obtained upon multiplying Eq. (2.23) by $2^k k!$,

$$\begin{aligned} k(k-1) \left(2^k \times 0^{k-2} + 2^k \alpha \right) &= k \left[2(-1)^{k-1} a \right] + \\ &+ \left[0^k b + 2^k c + 4^k d + 6^k e \right], \end{aligned}$$

and expanding it in matrix form,

$$\begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 2 & 0 & 2 & 4 & 6 \\ -8 & -4 & 0 & 4 & 16 & 36 \\ -48 & 6 & 0 & 8 & 64 & 216 \\ -192 & -8 & 0 & 16 & 256 & 1296 \\ -640 & 10 & 0 & 32 & 1024 & 7776 \end{pmatrix} \times \begin{pmatrix} \alpha \\ a \\ b \\ c \\ d \\ e \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 8 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

The solution is

$$\begin{pmatrix} \alpha \\ a \\ b \\ c \\ d \\ e \end{pmatrix} = \begin{pmatrix} -979/71 \\ -160/71 \\ -3931/213 \\ 2543/71 \\ -1315/71 \\ 247/213 \end{pmatrix}.$$

2.6 The algorithm to compute the coefficients

In Sections 2.5.1, 2.5.2 and 2.5.3, three compact finite difference schemes have been presented — Eqs. (2.18), (2.20) and (2.22) — and their respective coefficients have been determined by solving the system of equations originating from each — Eqs. (2.19), (2.21) and (2.23) —, using MATLAB's Symbolic Toolbox for the actual computation.

In the meanwhile, some hints have been given about the one-to-one link between the finite difference equations, and the corresponding set of k -indexed equations that must be solved for the coefficients. The understanding of such a link allows one to unambiguously write down the latter algebraic system as a simple transform of the former difference equation, with no need to explicitly expanding its terms in Taylor series.

A finite difference scheme — being it compact or not — is, at the highest possible level of interpretation, a linear combination of values of a function and/or its derivatives of various orders, located at several points. Some of these values are considered unknowns, and must be determined based on the remaining ones. Of course, the linear combination holds true up to a truncation error, which vanishes as some power of h . The general difference equation for which we seek the non-dimensional coefficients α_j ,

can be cast in the following form,

$$\sum_{j=0}^n \alpha_j h^{d_j} f_{r_j}^{(d_j)} = \mathcal{O}(h^p), \quad (2.24)$$

where $f_{r_j}^{(d_j)}$ is the derivative of order d_j of f , located at the point x_j , such that $r_j = \frac{x_j - x_0}{h}$, with x_0 a reference point and h a reference length (e.g., the mesh constant or average spacing). The truncation error is explicitly shown, and its order p is determined in the following as well. It is worth to stress, however, that p is the degree of the leading term of the truncation error, not the order of accuracy of the scheme. Indeed, if a derivative of order \tilde{d} is considered unknown in Eq. (2.24), then the order of accuracy with respect to *that* variable is $p - \tilde{d}$ (see Section 2.9.3 for further details). Eq. (2.24) involves, for each mesh point j , a derivative of order d_j located at x_j , which is non-dimensionalized by the appropriate power of the length h multiplying it. In this frame the ratios r_j play the role of the indices of the knowns and unknowns of the previous examples.⁶

By expanding the general term of the summation in Taylor series about x_0 , one obtains,

$$\sum_{j=0}^n \alpha_j h^{d_j} \sum_{i=0}^{+\infty} f_{r_0}^{(d_j+i)} \frac{r_j^i h^i}{i!} = \mathcal{O}(h^p).$$

Taking $\alpha_j h^{d_j}$ in the inner summation, and doing the change of variable $i \mapsto i = d_j + i$, the left hand side becomes

$$\sum_{j=0}^n \sum_{i=d_j}^{+\infty} \alpha_j f_{r_0}^{(i)} \frac{r_j^{i-d_j} h^i}{(i-d_j)!}.$$

On account of the notation of convenience concerning the factorial at the denominator, already adopted in Section 2.5.2, the lower bound of the inner summation can be set to $-\infty$, and the two summations can be switched, resulting in

$$\sum_{i=-\infty}^{+\infty} f_{r_0}^{(i)} h^i \sum_{j=0}^n \alpha_j \frac{r_j^{i-d_j}}{(i-d_j)!}.$$

⁶Indeed, if the mesh is uniform, than an obvious choice for the reference length h , is the constant spacing; in this case $r_j \in \mathbb{Z}$, just like was the case in several examples presented so far.

Requiring that this quantity be at least of order $\mathcal{O}(h^n)$ results in the following homogeneous $n \times (n + 1)$ system of equations,

$$\sum_{j=0}^n \frac{r_j^{i-d_j}}{(i-d_j)!} \alpha_j = 0 \quad \text{for } i = 0, 1, \dots, n-1.$$

To avoid, the indeterminacy of the solution, $\alpha_c \doteq 1$ is set for a single term of index c , and take it to the RHS,

$$\sum_{\substack{j=1 \\ j \neq c}}^n \frac{r_j^{i-d_j}}{(i-d_j)!} \alpha_j = -\frac{r_c^{i-d_c}}{(i-d_c)!} \quad \text{for } i = 0, 1, \dots, n-1, \quad (2.25)$$

thus obtaining an inhomogeneous square system of equations for the n remaining coefficients α_j , which can be concisely cast in matrix form as

$$\mathbf{M}\boldsymbol{\alpha} = \mathbf{q}, \quad (2.26)$$

with obvious meaning of the symbols. \mathbf{M} is a Vandermonde-like matrix, whose general element $m_{i,j}$ can be immediately determined, as much as the element q_i of \mathbf{q} , based on the finite difference Eq. (2.24). Indeed, each column of \mathbf{M} , as well as the column vector \mathbf{q} , originates from one addend of Eq. (2.24), according to the following transformation,

$$h^{d_j} f_{r_j}^{(d_j)} \longleftrightarrow \frac{r_j^{i-d_j}}{(i-d_j)!}, \quad \text{for } i = 0, 1, \dots, n-1. \quad (2.27)$$

Whether \mathbf{M} is singular or not, depends on the set of mesh points $\{x_j\}$, as well as on the order of differentiation $\{d_j\}$ of the quantity prescribed at each point. (The stencil of a typical ill-posed scheme is reported in Fig. 2.10a.) If \mathbf{M} is non-singular, the system Eq. (2.26) can be solved arithmetically, and its solution plugged in Eq. (2.24), thus obtaining the sought finite difference equation, whose truncation error is at least of order n .

The two equivalent functions reported in Listings A.1 and A.2, take in input the sets $\{r_j\}$ and $\{d_j\}$, as well as the index c used to impose $\alpha_c = 1$, and exploit the relation in Eq. (2.27), to build the system Eq. (2.26) and solve it for the α_j coefficients.

2.7 Hermitian compact schemes

As a side work, the author developed a new class of staggered compact differentiation and interpolation operators. The algorithm has its roots in an implicit interpolation theory consistent with compact schemes [52] and reduces to the computation of the required staggered derivatives and interpolated quantities as a combination of nodal values and collocated compact derivatives.

For instance, Eq. (2.3) can be used to determine the collocated first derivatives on the whole mesh, and the values obtained $\{f'_j\}_j$, together with the original set $\{f_j\}_j$, can be used to feed Hermitian schemes for the interpolant, staggered first derivative, and collocated first derivative,

$$\begin{aligned} f'_{j+1/2} &= 3 \frac{f_{j+1} - f_j}{2h} - \frac{f'_j + f'_{j+1}}{4}, \\ f_{j+1/2} &= \frac{1}{2}(f_j + f_{j+1}) + \frac{h}{8}(f'_j - f'_{j+1}), \\ f''_j &= 2 \frac{f_{j-1} - 2f_j + f_{j+1}}{h^2} - \frac{f'_{j+1} - f'_{j-1}}{2h}, \end{aligned}$$

This new approach is cost-effective, simplifies the imposition of boundary conditions, and has improved spectral resolution properties, on equal order of accuracy, with respect to classical schemes, as proved in [51].

2.8 Conservative compact schemes

Let the following be a compact scheme for the first derivative

$$\mathbf{A} \mathbf{f}' = \mathbf{B} \mathbf{f}, \quad (2.28)$$

where asymmetric schemes are used in the first and last few rows of \mathbf{A} and \mathbf{B} to impose boundary conditions. Furthermore assume that, for the sake of simplicity, the mesh is uniform, so that a central scheme is used in other equations.

Multiplying both sides of Eq. (2.28) by a diagonal matrix \mathbf{W} , does not affect its solution, so the scheme can be re-cast by setting $\tilde{\mathbf{A}} = \mathbf{W} \mathbf{A}$ and $\tilde{\mathbf{B}} = \mathbf{W} \mathbf{B}$,

$$\tilde{\mathbf{A}} \mathbf{f}' = \tilde{\mathbf{B}} \mathbf{f}. \quad (2.29)$$

At this point, the compact operator is required to be telescopic, i.e., that the contribution to the LHS only depends on first and last elements of \mathbf{f} . To do so, both sides of Eq. (2.29) are integrated through a product by the row vector $\mathbf{1}^T$ (the integral operator with all weighs set to 1), and require that

$$\mathbf{1}^T \tilde{\mathbf{B}} = \mathbf{1}^T \mathbf{W} \mathbf{B} = (-1, 0, 0, \dots, 0, 0, +1). \quad (2.30)$$

Eq. (2.30) is the system for the elements of \mathbf{W} ; indeed, the column vector \mathbf{w} can be defined such that $\mathbf{1}^T \mathbf{W} = \mathbf{w}^T$, and Eq. (2.30) can be cast as

$$\mathbf{w}^T \mathbf{B} = (-1, 0, 0, \dots, 0, 0, +1). \quad (2.31)$$

Since a central symmetric scheme is used in most points, the columns of \mathbf{B} not involved in boundary conditions sum up to 0, so the solution \mathbf{w} mostly contains ones, except for a few leading and trailing elements.

With the weighs \mathbf{w} determined, and the solution of Eq. (2.28) being $\mathbf{f}' = \mathbf{A}^{-1} \mathbf{B} \mathbf{f}$, the discrete integral of \mathbf{f}' through the quadrature coefficients defined as

$$\mathbf{s} = \mathbf{w}^T \mathbf{A} \quad (2.32)$$

correctly results dependent only on boundary elements of \mathbf{f}

$$\mathbf{w}^T \mathbf{A} \mathbf{f}' = f_{\text{last}} - f_{\text{first}}.$$

A paper by Knikker [24] inspired this argument in favour of the following conclusion. As long as Eq. (2.31) admits a solution, the compact scheme of Eq. (2.28) is conservative with respect to the quadrature weights \mathbf{s} defined in Eq. (2.32), and the weights \mathbf{w} , solution of Eq. (2.31), need not be used explicitly [25], since Eqs. (2.28) and (2.29) coincide.

This conclusion can be extended to compact schemes for successive derivatives [14].

2.9 Further considerations

2.9.1 Ill-posed schemes

An example of stencil related to an ill-posed scheme is depicted in Fig. 2.10a. The reason is easily understood: since the values prescribed are in number of three, the function passing through them should be a second degree polynomial, i.e., a parabola, whose slope in the middle point is well-known to be unconditionally equal to the average slope from the

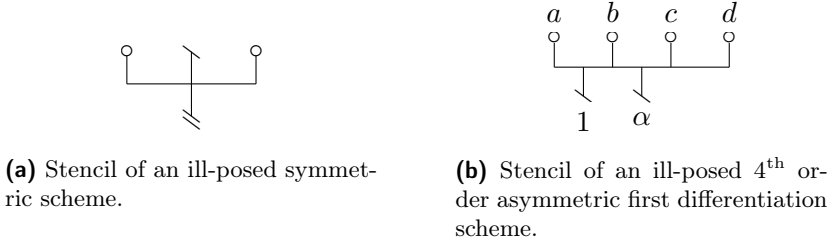


Figure 2.10: Stencils of ill-posed schemes.

left point to the right, and cannot be assigned independently; assigning it results either in no parabola at all, or in infinite parabola with varying curvatures, hence the ill-posedness of the scheme.

Such a scheme is an purely illustrative, and it would be unlikely met in practical applications. A case which is more likely encountered has the stencil depicted in Fig. 2.10b; the system for the coefficients is the following,

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ -2 & -1 & 1 & 3 & 5 \\ -8 & 1 & 1 & 9 & 25 \\ -24 & -1 & 1 & 27 & 125 \\ -64 & 1 & 1 & 81 & 625 \end{pmatrix} \times \begin{pmatrix} \alpha \\ a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad (2.33)$$

which is inconsistent, being 4 the rank of the matrix.⁷

In order to gain deeper knowledge, it is worthwhile to further investigate Eq. (2.33). The rank of the augmented matrix is 5, so there exists *no* linear combination of the columns of \mathbf{M} that equals the RHS. This means that the vector \mathbf{q} and one column of \mathbf{M} (e.g., the first) can be switched, and obviously changed in sign, to get a determined system, $\widetilde{\mathbf{M}}\widetilde{\boldsymbol{\alpha}} = \widetilde{\mathbf{q}}$. Translating this action on the stencil of Fig. 2.10b, the collocation point (labeled as 1) and the point labeled as α can be switched to obtain a 4th order scheme. Still the vector \mathbf{q} , that now multiplies α , is not spanned by the columns of \mathbf{M} , so that it must be $\alpha = 0$, thus yielding the well-known 4th order explicit scheme.

⁷The non-void kernel is spanned by the eigenvector $[-24, -1, 27, -27, 1]^T$.

2.9.2 Parametric schemes

The inconsistency of Eq. (2.33) does not mean that the stencil of Fig. 2.10b cannot be used if not changing the collocation point; indeed, we can give up on 4th order accuracy, cutting out the last equation, and taking one coefficient (e.g., α) to the RHS, thus obtaining the system for the coefficients of a one-parameter family of 3rd order schemes, which is used with $\alpha = 0$ in [25]:

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & 3 & 5 \\ 1 & 1 & 9 & 25 \\ -1 & 1 & 27 & 125 \end{pmatrix} \times \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 0 \\ 2 + 2\alpha \\ 8\alpha \\ 24\alpha \end{pmatrix},$$

whose solution is

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \frac{1}{24} \begin{pmatrix} \alpha - 23 \\ 21 - 27\alpha \\ 27\alpha + 3 \\ -\alpha - 1 \end{pmatrix}.$$

2.9.3 The symmetry and the order of accuracy

It is worth observing that, once the coefficients α_j are determined by solving Eq. (2.26) and plugged into Eq. (2.24), some leading terms of the truncation error can end up being zero already, as a consequence of the mutual positions where the various terms are located. In short, depending on $\{x_j\}$ and $\{d_j\}$, it can automatically result that $p > n$ in Eq. (2.24). In fact it can be either n or $n+1$. Indeed, since in the case \mathbf{M} is non-singular, and its rows are linearly independent, the equations relative to $i \geq n$ in Eq. (2.25) must be a linear combination of the preceding n . If the element q_i is zero, then the order of the truncation error is higher.

A notable situation in which this happens, occurs when symmetric schemes are concerned. In this case, having labelled the central point as x_0 and used it for the Taylor expansion, the columns of \mathbf{M} relative to symmetric points of the stencil have the same absolute values, but one has constant sign, whereas the other has alternating signs. The coefficients relative to those columns can be appropriately set equal or opposite to each other, and those columns can be summed or subtracted; the resulting null rows can be removed, as they represent equalities holding by virtue of symmetry. This concept is better explained by the following example, which is the occasion for another interesting comment.

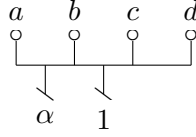


Figure 2.11: The same stencil as in Fig. 2.10a, except for the point whose coefficient is set to 1.

The coefficients of the scheme with the stencil depicted in Fig. 2.11, which is asymmetric, fulfill the following system,

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ -1 & -3 & -1 & 1 & 3 \\ 4 & 9 & 1 & 1 & 9 \\ -12 & -27 & -1 & 1 & 27 \\ 32 & 81 & 1 & 1 & 81 \end{pmatrix} \times \begin{pmatrix} \alpha \\ a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad (2.34)$$

whose solution is

$$\begin{pmatrix} \alpha \\ a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 0 \\ 1/48 \\ -9/16 \\ 9/16 \\ -1/48 \end{pmatrix}. \quad (2.35)$$

The result that $\alpha = 0$ testifies that, when only one point breaks the symmetry of a scheme, then its coefficient must be zero, if the maximum order is requested.

The system (2.34) can be then rewritten with the last equation and the first column and unknown α removed, since the latter is known to be 0,

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ -3 & -1 & 1 & 3 \\ 9 & 1 & 1 & 9 \\ -27 & -1 & 1 & 27 \end{pmatrix} \times \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}.$$

It is quite clear that first and third equations become identities $0 = 0$ by setting $a = -d$ and $b = -c$, and, as such, can be removed; the columns can be subtracted accordingly, yielding

$$\begin{pmatrix} 2 & 6 \\ 2 & 54 \end{pmatrix} \times \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix},$$

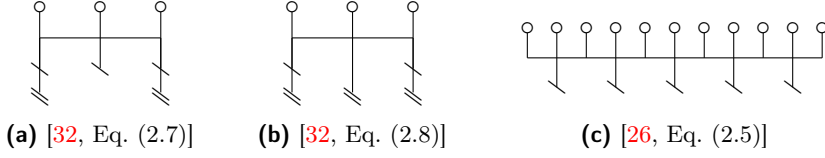


Figure 2.12: (a) and (b): sixth order CCD scheme for first and second derivatives, [32] (note that in the left and right points both first and second unknown derivatives are located); (c): CCS scheme for first derivatives, [26] (note that both cell-centered and face-centered values are used at the RHS).

whose solution is still Eq. (2.35).

As a last remark, symmetry must be identified by considering knowns and unknowns together. After all, the distinction only consists in choosing those terms of Eq. (2.24) that are taken to the RHS and those kept in the LHS. In this respect, some schemes, apparently different and presented as such, are in fact the same exact equation with a different selection of unknowns [24, Eq. (20) and (21)].

2.9.4 CCD and other schemes

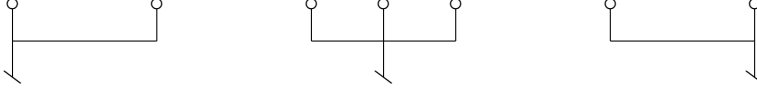
The 6th order CCD schemes [32] merely use the same coordinate for both the value of first and second derivatives in Eq. (2.24), and consider them unknown. The stencil relative to those schemes, depicted in Figs. 2.12a and 2.12b, is explanatory in this respect. The coefficients of such CCD schemes can be determined with the procedure outlined in Section 2.6.

The same holds for the CCS schemes presented more recently [26], whose stencil is depicted in Fig. 2.12c.

2.9.5 Compact schemes as linear combinations of explicit finite differences

Eqs. (2.1) and (2.3) are very similar in their essence, in that both represent a relation between nodal values and nodal derivatives. This relation is expressed by a linear combination of those values, through coefficients to be determined by imposing that the truncation error be of the desired degree (which is limited by the number of available coefficients).

Compact schemes can be also obtained as linear combinations of explicit finite difference schemes, as is done for the Padé scheme in Section 2.9.5.



(a) Stencil of Eq. (2.37). (b) Stencil of Eq. (2.36). (c) Stencil of Eq. (2.38).

Figure 2.13: Stencils of Eqs. (2.36) to (2.38).

Padé scheme revisited

The truncation error of Eq. (2.8) can be shown to be $-y'''h^2/3!$, so that equation can be rewritten for convenience as

$$y'_j = \frac{y_{j+1} - y_{j-1}}{2h} - \left[y''' \frac{h^2}{3!} + y^{(5)} \frac{h^4}{5!} + \dots \right] \quad (2.36)$$

The forward and backward versions of Eq. (2.36) for $y'_{j\pm 1}$, based on the same two values, are

$$y'_{j-1} = \frac{y_{j+1} - y_{j-1}}{2h} - \left[y''_j h - y'''_j \frac{h^2}{3} + y^{(4)}_j \frac{h^3}{3!} - y^{(5)}_j \frac{h^4}{30} + \dots \right] \quad (2.37)$$

$$y'_{j+1} = \frac{y_{j+1} - y_{j-1}}{2h} + \left[y''_j h + y'''_j \frac{h^2}{3} + y^{(4)}_j \frac{h^3}{3!} + y^{(5)}_j \frac{h^4}{30} + \dots \right] \quad (2.38)$$

All three schemes are depicted in Fig. 2.13 accordingly to the convention established in Section 2.2, and it is natural to argue that the three equations can be linearly combined in order to eliminate further terms of the truncation error; specifically, it is easy verifiable that multiplying Eqs. (2.37) and (2.38) by $1/4$ and adding them both to Eq. (2.36), the Padé scheme is recovered, together with its truncation error.

It is worth to note that Eqs. (2.37) and (2.38) are first order accurate and, as indicated by Figs. 2.13a and 2.13c, do not make use of the value y'_j , whereas Eq. (2.36) does use it — hence the central circle in Fig. 2.13b —, but the coefficient happens to be zero by symmetry. Should the central point be included in the two asymmetric formulæ, Eqs. (2.37) and (2.38) would have been second order accurate, but the overall combination of the three would have invariably led to the Padé scheme.

2.10 Impact of boundary schemes over global accuracy

In the present section an accuracy analysis is conducted to assess how much the accuracy granted by the use of a symmetric compact scheme is compromised by the asymmetric compact schemes used at the boundaries.

A 6th order interior scheme has been chosen for interpolation, first and second differentiation. To retain the compactness of the schemes as much as possible, the boundary formulæ are chosen to be compact too, but in order to keep the matrices **A** of the three schemes tridiagonal, they can involve only the first two unknowns (on the left boundary, the last two on the right one). This causes the bandwidth of **B** matrices to increase significantly if the same order of interior is required to boundary schemes. The 6th order interior scheme has been coupled to 4th, 5th and 6th order boundary closure. All of them use the stencils collected in Table 2.1. It can be noted that two of them are quite unusual in that a node has been shifted, the reason being explained in the caption.

In Fig. 2.15 it can be seen that the global order of accuracy of a compact method — interpolation as well as differentiation of both first and second order — is indeed influenced by boundary schemes' accuracy, as compared to that of interior schemes.

- In spite of internal 6th order scheme, the accuracy of the method is driven down to 4th order when a 4th order boundary scheme is used. This happens in all the three cases analyzed (i.e., interpolation, first and second differentiation);
- when the boundary scheme is 6th order accurate — i.e., it equals internal accuracy — the maximum order is preserved in all the three cases;
- when the boundary scheme is 5th order accurate, some differences depend on whether the scheme is for interpolation, first or second differentiation:
 - *interpolation*: internal accuracy is almost unaffected when $N < 500$, whereas it is diminished by 1 if $N > 500$;
 - *first derivative*: for $N > 200$ the accuracy is clearly 5th order;
 - *second derivative*: the accuracy is not affected at all, before round-off errors occur.

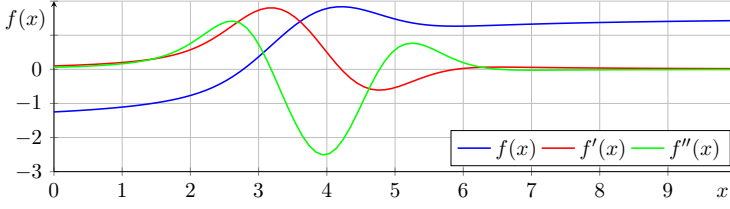


Figure 2.14: Test-function (Eq. (2.39)) and its first and second derivatives.

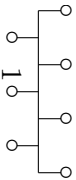
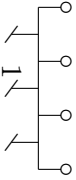
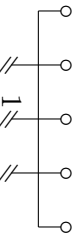
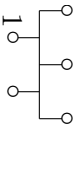
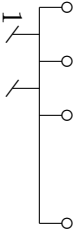
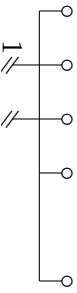
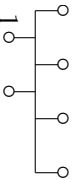
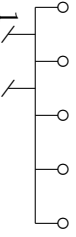
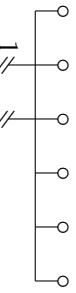
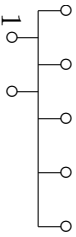
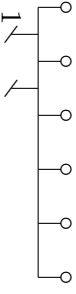
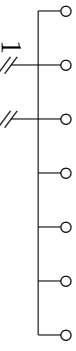
Other two observations are of interest:

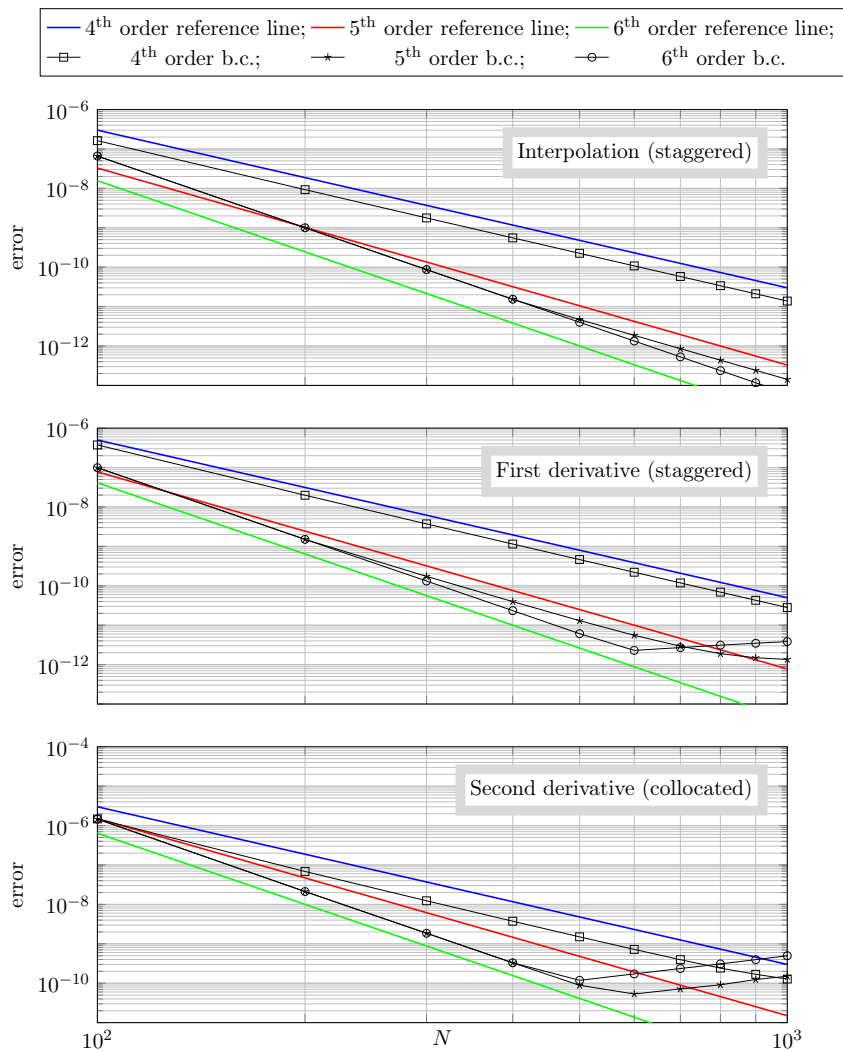
- the analysis is not significant both when N is too low (say $N < 200$) and when round-off errors become important (not occurred for interpolation, $N > 600$ for first derivative, $N > 500$ for second derivative);
- away from the boundaries, the inaccuracy of boundary schemes does not compromise that granted, in the interior part of the domain, by high order symmetric formulæ.

Obviously the analysis is dependent on the test function the schemes are applied to. In the present work the following test function has been used:

$$f(x) = \arctan(x - 3) + e^{-(x-4)^2}. \quad (2.39)$$

Table 2.1: Stencils used in the analysis whose results are plotted in Fig. 2.15. Collocation nodes are labeled as 1. Note that 4th order schemes for first and second differentiation are anomalous, in that the most external node has been shifted further away by a distance h in order to overcome the inconsistency of the system of equations for the coefficients discussed in Section 2.9.

order	interpolant	first derivative	second derivative
6 th			
4 th			
5 th			
6 th			

**Figure 2.15:** Results of the accuracy analysis.

Chapter 3

Parallelization of linear banded systems

Let $\varphi(x)$ be a function of x defined on a interval, and $\varphi_i = \varphi(x_i)$ the values that φ assumes on a certain mesh whose points are x_i with $i = 1, 2, \dots, N$, and be these values collected in the numerical vector φ .

Denoting with a prime the operation of differentiation (being it exact or not) and referring to an explicit finite difference approach, the problem of determining the approximate values of the derivative of φ in the points x_i reduces to the simple matrix product

$$\varphi' = \mathbf{D}\varphi, \quad (3.1)$$

being \mathbf{D} an $N \times N$ differentiation matrix (usually sparse). In this case the relation between φ' and φ is explicit, i.e., each component of the vector φ' can be computed in parallel, being known both the matrix \mathbf{D} and the vector φ . The product at the RHS of Eq. (3.1) can be easily parallelized.

The use of implicit schemes (i.e., compact schemes) leads instead to the problem

$$\mathbf{A}\varphi' = \mathbf{B}\varphi, \quad (3.2)$$

in which the matrix \mathbf{A} is a sparse matrix (tridiagonal, pentadiagonal, etc). Note that Eq. (3.1) can be seen as a special case of Eq. (3.2) resulting from the definition $\mathbf{B} \doteq \mathbf{D}$ and the choice $\mathbf{A} = \mathbf{I}$, being \mathbf{I} the identity matrix. Again, the computation of the RHS is a simple matter of parallelizing a matrix product, so the result can be written as $\mathbf{q} \doteq \mathbf{B}\varphi$. On the other hand, the relation between φ' and φ is implicit, so the computation of φ'

requires now the resolution of the following linear system,

$$\mathbf{A}\boldsymbol{\psi} = \mathbf{q}, \quad (3.3)$$

where $\boldsymbol{\psi} \doteq \boldsymbol{\varphi}'$. Eventually, the task becomes the parallel resolution of the sparse linear system in which \mathbf{A} is a $N \times N$ sparse matrix, $\boldsymbol{\psi}$ is the $N \times 1$ unknown column vector, and \mathbf{q} is the $N \times 1$ column vector of knowns. Moreover the focus is on linear *banded* systems, in which \mathbf{A} is a banded matrix.

3.1 Exact parallelization

In this section the case of a tridiagonal system is analyzed. The global system is split up into a number of sub-systems, each of which is coupled to the preceding and following by one equations each.

In order to provide enough understanding of the ideas and problems underlying the parallelization, the system is first split up in two, and the sub-systems and the two scalar coupling equations are treated explicitly. In a second moment a general number of sub-systems considered, starting over from another, more convenient, point of view.

3.1.1 Two processors

Here the assumption is made that two processes are available for the solution of a tridiagonal system, whose matrix is \mathbf{A} , the extension to multi-diagonal system being straight forward, although requiring a careful inspection of what are referred to as “coupling coefficients” in the following.

The unknown vector $\boldsymbol{\psi}$ is distributed among two processes, i.e., one process handles its first M components, whereas the other process handles the remaining $N - M$, that is

$$\boldsymbol{\psi}^{(1)} = [\psi_1^{(1)}, \psi_2^{(1)}, \dots, \psi_M^{(1)}]^T \doteq [\psi_1, \psi_2, \dots, \psi_{M-1}, \psi_M]^T \quad (3.4)$$

$$\boldsymbol{\psi}^{(2)} = [\psi_1^{(2)}, \psi_2^{(2)}, \dots, \psi_{N-M}^{(2)}]^T \doteq [\psi_{M+1}, \psi_{M+2}, \dots, \psi_{N-1}, \psi_N]^T. \quad (3.5)$$

where $\boldsymbol{\psi}^{(1)}$ and $\boldsymbol{\psi}^{(2)}$ are the process-local sub-vectors. Likewise, the known vector \mathbf{q} is distributed to the two processes, resulting in the sub-vectors $\mathbf{q}^{(1)}$ and $\mathbf{q}^{(2)}$. The two processes have to solve Eq. (3.3) for their own vectors of unknowns, thus a proper sub-matrix of \mathbf{A} must be available

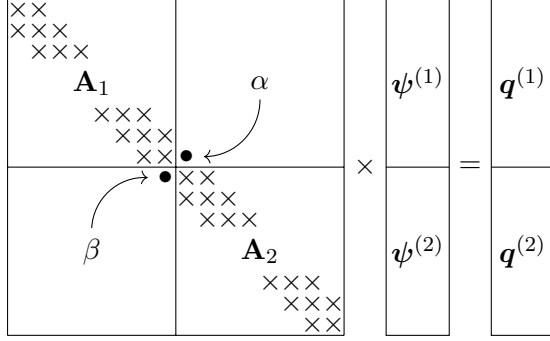


Figure 3.1: Block structure of the system (3.3): matrix \mathbf{A} seen as a 2×2 -block matrix, vectors $\boldsymbol{\psi}$ and \mathbf{q} as 2×1 -block vectors. First and last equations, i.e., rows of \mathbf{A} , must be boundary conditions, so they can have more than two non-zero elements, contrary to what is sketched.

to each. To understand which blocks of the matrix each process need to access to, it is useful to sketch a simple scheme illustrating the layout of Eq. (3.3).

As depicted in Fig. 3.1, if the elements α and β were zero, the matrix \mathbf{A} would be a block-diagonal matrix, thus allowing a perfect parallel resolution of the system (3.3), but this is not the case, in the sense that these “coupling coefficients” α and β are not zero in general.¹ The term α multiplies $\psi_1^{(2)}$, the first component of $\boldsymbol{\psi}^{(2)}$, whereas β multiplies $\psi_M^{(1)}$, the last component of $\boldsymbol{\psi}^{(1)}$; so the original $N \times N$ system (3.3) can be split in two as follows,

$$\begin{cases} \mathbf{A}_1 \boldsymbol{\psi}^{(1)} + \alpha \psi_1^{(2)} \boldsymbol{\delta}_M^{(1)} = \mathbf{q}^{(1)} \\ \mathbf{A}_2 \boldsymbol{\psi}^{(2)} + \beta \psi_M^{(1)} \boldsymbol{\delta}_1^{(2)} = \mathbf{q}^{(2)} \end{cases} \quad (3.6)$$

where the M^{th} (i.e., last) and 1^{st} columns of the identity matrices of size $M \times M$ and $(N - M) \times (N - M)$ are denoted by $\boldsymbol{\delta}_M^{(1)}$ and $\boldsymbol{\delta}_1^{(2)}$ respectively. Eq. (3.6) can be formally solved for $\boldsymbol{\psi}^{(1)}$ and $\boldsymbol{\psi}^{(2)}$ taking coupling terms to the RHS and inverting the blocks \mathbf{A}_1 and \mathbf{A}_2 independently,

$$\begin{cases} \boldsymbol{\psi}^{(1)} = -\alpha \psi_1^{(2)} \mathbf{A}_1^{-1} \boldsymbol{\delta}_M^{(1)} + \mathbf{A}_1^{-1} \mathbf{q}^{(1)} \\ \boldsymbol{\psi}^{(2)} = -\beta \psi_M^{(1)} \mathbf{A}_2^{-1} \boldsymbol{\delta}_1^{(2)} + \mathbf{A}_2^{-1} \mathbf{q}^{(2)}. \end{cases} \quad (3.7)$$

¹For the classic Padé scheme \mathbf{A} is tridiagonal with elements $[\frac{1}{4}, 1, \frac{1}{4}]$, so $\alpha = \beta = \frac{1}{4}$.

In the RHSs, vectors $\delta_M^{(1)}$ and $\delta_1^{(2)}$ extract the last column of \mathbf{A}_1^{-1} and the first of \mathbf{A}_2^{-1} respectively. These columns, namely \mathbf{a} and \mathbf{b} , can be computed in parallel and, most importantly, *una tantum*. The last terms of the RHSs of Eq. (3.7) are the solutions of the two uncoupled systems

$$\mathbf{A}_1 \psi^{0(1)} = \mathbf{q}^{(1)} \quad (3.8)$$

$$\mathbf{A}_2 \psi^{0(2)} = \mathbf{q}^{(2)}, \quad (3.9)$$

which can be evaluated in parallel. With these definitions the solution Eq. (3.7) can be rewritten in a clearer form,

$$\begin{cases} \psi^{(1)} = -\alpha \psi_1^{(2)} \mathbf{a} + \psi^{0(1)} \\ \psi^{(2)} = -\beta \psi_M^{(1)} \mathbf{b} + \psi^{0(2)} \end{cases} \quad (3.10a)$$

$$(3.10b)$$

At this point, coupling terms are the ones multiplying α and β , and they can be determined as follows. The last row of Eq. (3.10a) and first one of Eq. (3.10b) are extracted, thus resulting in the 2×2 -scalar system

$$\begin{cases} \psi_M^{(1)} = -\alpha \psi_1^{(2)} a_M + \psi_M^{0(1)} \\ \psi_1^{(2)} = -\beta \psi_M^{(1)} b_1 + \psi_1^{0(2)}, \end{cases} \quad (3.11)$$

whose solutions are

$$\begin{cases} \psi_M^{(1)} = \frac{\psi_M^{0(1)} - \alpha a_M \psi_1^{0(2)}}{1 - \alpha \beta a_M b_1} \\ \psi_1^{(2)} = \frac{\psi_1^{0(2)} - \beta b_1 \psi_M^{0(1)}}{1 - \alpha \beta a_M b_1}. \end{cases} \quad (3.12)$$

These values can be inserted into Eq. (3.10), to obtain the final solutions of Eq. (3.6), that is, the solution of the original system (3.3).

The process of solution of the $N \times N$ system (3.3) as obtained by two processes can be summarized as follows:

1. M unknowns are assigned to the first process and the remaining $N - M$ unknowns to the second process;
2. the first process extracts the last column of \mathbf{A}_1^{-1} , the second process the first one of \mathbf{A}_2^{-1} , in parallel;
3. solve Eqs. (3.8) and (3.9) for $\psi^{0(1)}$ and $\psi^{0(2)}$, respectively;
4. $\psi_M^{(1)}$ and $\psi_1^{(2)}$ are evaluated through Eq. (3.12);
5. $\psi^{(1)}$ and $\psi^{(2)}$ are evaluate through Eq. (3.10).

3.1.2 Multi processors

In the previous section the problem of parallelizing the solution of the system $\mathbf{A}\psi = \mathbf{q}$ has been dealt with by manipulating directly the equations. Fig. 3.1 has been presented just to facilitate the interpretation of the equations.

In the present section the problem of the parallelization is extended to the case that n processes are available, which is not straightforward if dealing directly with the equations, like in the previous section; in fact, the general form of the coupling system is not even embodied by Eq. (3.11). For this reason it is worthwhile to approach the problem by analyzing the block structure of the system $\mathbf{A}\psi = \mathbf{q}$, depicted in Fig. 3.2, and then solving it as a whole, without explicitly considering the single blocks.

Being ψ unknown and \mathbf{q} arbitrary, there is nothing to exploit about their shape, so they can be simply split up into n sub-vectors each sized $M_i \times 1$. Obviously $\sum_{i=1}^n M_i = N$, but it is useful in the following to define $N_i \doteq \sum_{k=1}^i M_k$, the global index of the last term of the i^{th} block.

The tridiagonal matrix \mathbf{A} is symbolically split up in $n \times n$ blocks, the i^{th} diagonal block being a tridiagonal $M_i \times M_i$ sub-matrix; the terms denoted by the symbol \bullet are coupling coefficients between adjacent blocks. Fig. 3.2 suggests the obvious decomposition of the matrix \mathbf{A} as the following sum

$$\mathbf{A} = \mathbf{A}^{\text{BD}} + \mathbf{A}^{\text{OD}},$$

where the superscript BD denotes the Block-Diagonal part of \mathbf{A} , each block of which is a tridiagonal sub-matrix \mathbf{A}_i , whereas OD denotes the Off-block-Diagonal part, whose only non-zero elements are α_i and β_i . The latter, \mathbf{A}^{OD} , is better discussed later in this section.

With this decomposition the system can be rewritten as

$$[\mathbf{A}^{\text{BD}} + \mathbf{A}^{\text{OD}}]\psi = \mathbf{q}, \quad (3.13)$$

which almost corresponds to Eq. (3.6) in the preceding section.

Now Eq. (3.13) is formally multiplied by $(\mathbf{A}^{\text{BD}})^{-1}$, which is equivalent to multiply, for $i = 1, 2, \dots, n$, the i^{th} block of equations by the matrix \mathbf{A}_i^{-1} — which is invertible provided \mathbf{A} is diagonally dominant by rows — thus obtaining

$$[\mathbf{I} + (\mathbf{A}^{\text{BD}})^{-1} \mathbf{A}^{\text{OD}}]\psi = (\mathbf{A}^{\text{BD}})^{-1} \mathbf{q}; \quad (3.14)$$

that is, defining $\mathbf{S}^{(0)} \doteq (\mathbf{A}^{\text{BD}})^{-1} \mathbf{A}^{\text{OD}}$,

$$[\mathbf{I} + \mathbf{S}^{(0)}]\psi = (\mathbf{A}^{\text{BD}})^{-1} \mathbf{q}, \quad (3.15)$$

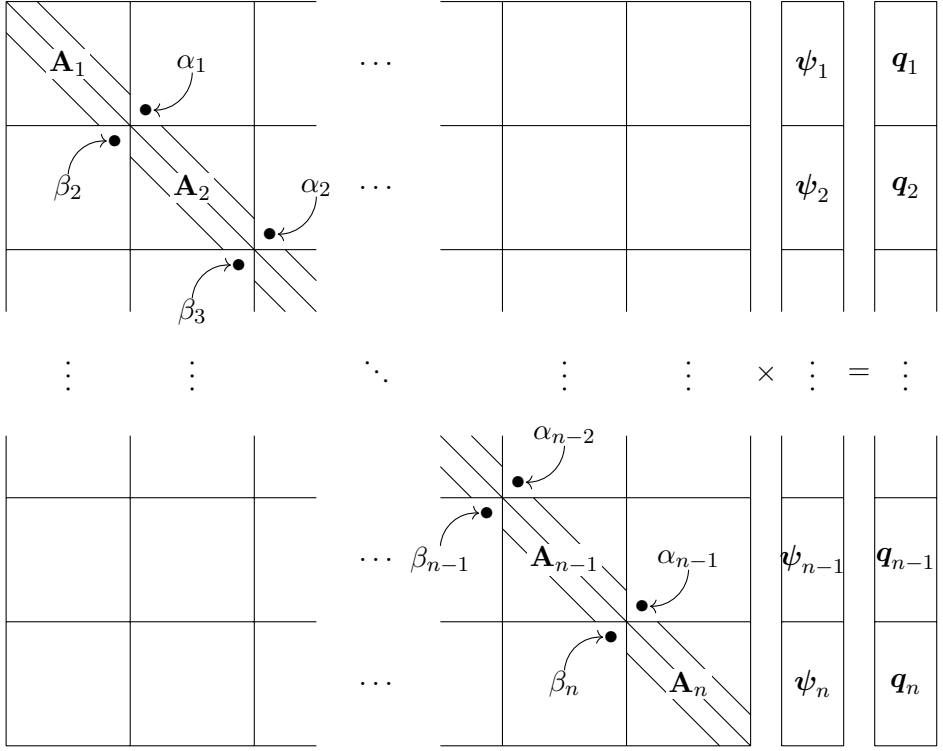


Figure 3.2: Block structure of the matrix \mathbf{A} , unknown vector ψ and known vector \mathbf{q} .

in which $(\mathbf{A}^{\text{BD}})^{-1}$ is still a block-diagonal matrix, whose general i^{th} block is in fact \mathbf{A}_i^{-1} . Actually Eq. (3.14) can be rewritten as

$$(\mathbf{A}^{\text{BD}})^{-1} \mathbf{A} \psi = (\mathbf{A}^{\text{BD}})^{-1} \mathbf{q}, \quad (3.16)$$

which makes evident that \mathbf{A}^{BD} takes the role of an block-diagonal algebraic preconditioner [57]. The RHS is a block-diagonal matrix times a block-column vector, that is, a block-column vector whose blocks are the vectors $\psi^{0(i)}$, solutions of n systems

$$\mathbf{A}_i \psi^{0(i)} = \mathbf{q}^{(i)}, \quad i = 1, 2, \dots, n, \quad (3.17)$$

which can be solved in parallel at each step of the iterative process. So it is convenient and straightforward to define the vector

$$\psi^0 \doteq (\mathbf{A}^{\text{BD}})^{-1} \mathbf{q}.$$

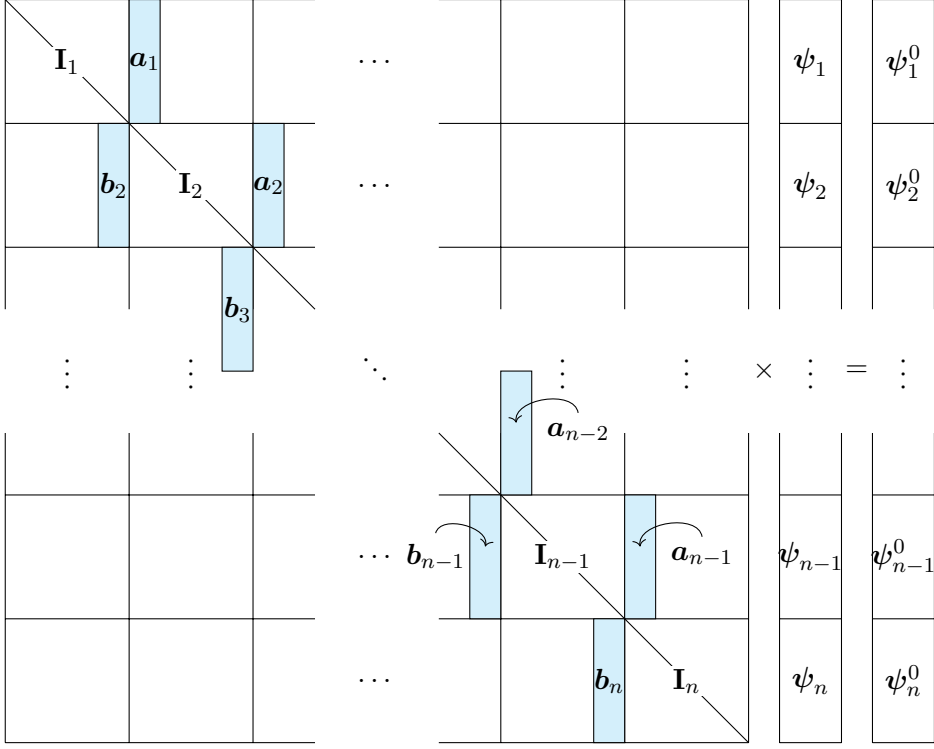


Figure 3.3: Block-structure of the spike matrix \mathbf{S} . Matrix $\mathbf{S}^{(0)}$ is obtained by substituting identity blocks by null blocks.

About the LHS, the matrix $\mathbf{S}^{(0)}$ is the product between the block-diagonal matrix $(\mathbf{A}^{\text{BD}})^{-1}$ and the matrix \mathbf{A}^{OD} , and deserves a better inspection with the help of Fig. 3.2 as follows: the i^{th} block-row of $\mathbf{S}^{(0)}$ is the product of \mathbf{A}_i^{-1} times the i^{th} block-row of \mathbf{A}^{OD} , the latter having only two non-null elements — β_i in the first row and α_i in the last row — that extract and multiply the first and last column of \mathbf{A}_i^{-1} respectively. These columns (already multiplied by β_i and α_i) are named \mathbf{b}_i and \mathbf{a}_i . Thus, assembling all these columns in their correct positions the matrix $\mathbf{S}^{(0)}$ is obtained, whose non-null elements are covered in cyan in Fig. 3.3. Upon adding the identity matrix \mathbf{I} to $\mathbf{S}^{(0)}$, the following \mathbf{S} matrix is obtained

$$\mathbf{S} \doteq \mathbf{I} + \mathbf{S}^{(0)} = \mathbf{I} + (\mathbf{A}^{\text{BD}})^{-1} \mathbf{A}^{\text{OD}},$$

which is the one depicted in Fig. 3.3, the observations of which, in conjunction with Fig. 3.2, clarifies also all the above description of the product

between $(\mathbf{A}^{\text{BD}})^{-1}$ and \mathbf{A}^{OD} . Note that \mathbf{S} is apparently an identity matrix with *spikes* protruding from its diagonal, so it is eloquently referred to as SPIKE matrix, and the whole algorithm is named *SPIKE algorithm* after it [17]. Note also that all these spikes, due to their definition, can be computed in parallel and *una tantum*; for instance, the i^{th} process, which workspace contains \mathbf{A}_i , can compute \mathbf{a}_i and \mathbf{b}_i independently from other processors.² Moreover, if $M_i = M = \text{const.}$ for all i and if the matrix \mathbf{A} is Toeplitz, as it is usually the case, the matrices \mathbf{A}_i are all equal (at least for $i = 2, 3, \dots, n-1$) and only two columns are required (at most four).

With the last two definitions of \mathbf{S} and ψ^0 , Eq. (3.15) can be succinctly rewritten as

$$\mathbf{S}\psi = \psi^0. \quad (3.18)$$

So far the system is not yet decoupled, but it the problem is not just shifted from an \mathbf{A} -system to \mathbf{S} -system. Indeed, a full coupled (or full implicit) system is transformed into a simpler one in which the “degree of coupling” is much lower. In the original problem, in fact, *each* equation was coupled with both the preceding and the following equations; in the new formulation, with the symbolic inversion of the block diagonal matrix

$$\underbrace{\mathbf{A}\psi = \mathbf{q}}_{\text{full coupling}} \xrightarrow{\text{inverting } \mathbf{A}^{\text{BD}}} \underbrace{\mathbf{S}\psi = \psi^0}_{\text{low coupling}},$$

a system is obtained, in which coupling exists only between the N_i^{th} and $(N_i + 1)^{\text{th}}$ equations for $i = 1, 2, \dots, n-1$. These equations are in number of $2(n-1) \doteq m$, and the sorted set of their indices is referred to as \mathbf{C} and, used as apex, indicates the sub-matrix or sub-vector obtained by retaining the rows of indices \mathbf{C} (and/or the columns, as the case may be).

So, in order to decouple the sub-systems, thus allowing a parallel resolution of Eq. (3.18), the “small” $m \times m$ determined sub-system involving only coupling unknowns (ψ_{N_i} and ψ_{N_i+1} for $i = 1, 2, \dots, n-1$) must be extracted from it. The coefficients, unknowns and knowns are collected into the matrix \mathbf{C} and the vectors $\psi^{\mathbf{C}}$ and $\psi^{0\mathbf{C}}$ respectively, all of them highlighted in red in Fig. 3.4.³ Upon this definition, the coupling system

²Obviously the first process does not need to compute \mathbf{b}_1 , whereas the last one does not need to compute \mathbf{a}_n .

³Maybe it is not wasteful to underline that \mathbf{C} is a matrix whereas \mathbf{C} is a set of indices used a superscript.

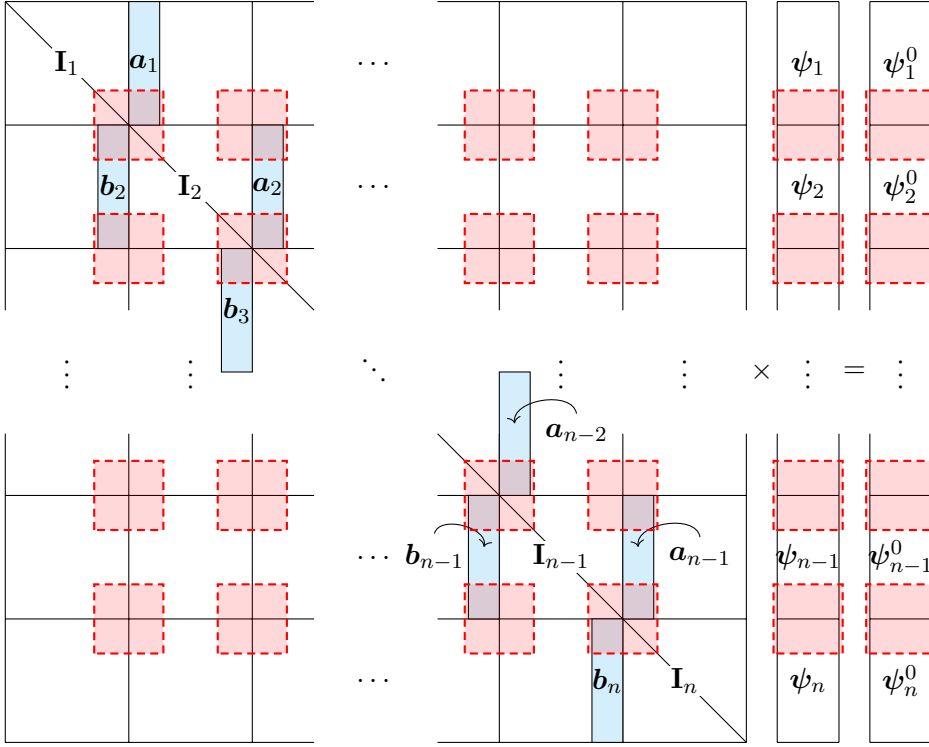


Figure 3.4: Block-structure of the matrix \mathbf{S} with coupling terms highlighted in red.

can be cast in the simple form:

$$\mathbf{C}\psi^C = \psi^{0C}. \quad (3.19)$$

This system is known in literature as *reduced system* and its matrix is often symbolized by $\hat{\mathbf{S}}$ [18, p. 500]. Fig. 3.4 shows only the corners of the matrix \mathbf{A} , and yet it is evident that \mathbf{C} is a sparse matrix too, whose diagonal is made up of ones. More precisely, the coupling matrix is a pentadiagonal matrix and its sparsity pattern, depicted in Fig. 3.5, shows that this coupling system is a block-tridiagonal system with 2×2 diagonal blocks.

At this point Eq. (3.19) can be solved for the coupling sub-vector ψ^C , whose components are the only ones affecting the matrix product $\mathbf{S}^{(0)}\psi$ in Eq. (3.15), rewritten here substituting the definition of ψ^0

$$\psi = -\mathbf{S}^{(0)}\psi + \psi^0, \quad (3.20)$$

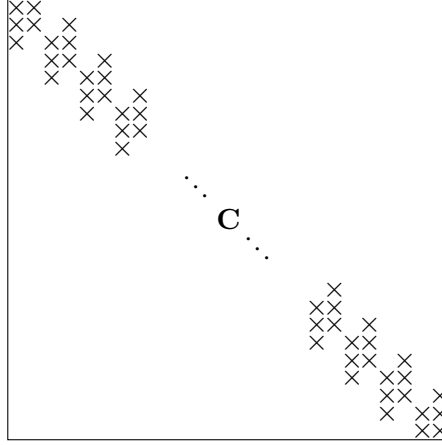


Figure 3.5: Sparsity pattern of the coupling matrix \mathbf{C} . \times -symbols on the diagonal are simply 1-s, while those out of diagonal depends on the particular matrix \mathbf{A} .

so that $\mathbf{S}^{(0)}$ can be “compacted”, i.e., its non-null columns are extracted and placed side-by-side to form an $N \times m$ matrix \mathbf{S}^C , and Eq. (3.20) rewritten as

$$\psi = -\mathbf{S}^C \psi^C + \psi^0 \quad (3.21)$$

becoming an explicit solution — at the time that ψ^C is available — which can be computed in parallel.⁴

Summarizing, in the context of the parallel solution of the $N \times N$ system (3.3) by means of n processes, each process does the following:

- it is assigned to M_i unknowns;
 - it extracts first and last columns of \mathbf{A}_i^{-1} , multiplying the first by β_i , the latter by α_i ;
1. it solves its own system (3.17) for $\psi^{0(i)}$;
 2. it sends $\psi_1^{0(i)}$ and $\psi_{M_i}^{0(i)}$ to a master process (solves the coupling system (3.19) for ψ^C) and receives back the elements $\psi_{M_{i-1}}^{(i-1)}$ and $\psi_1^{(i+1)}$;
 3. it computes $\psi^{(i)}$ through Eq. (3.21).

⁴This time the apex C is used to select columns instead of rows.

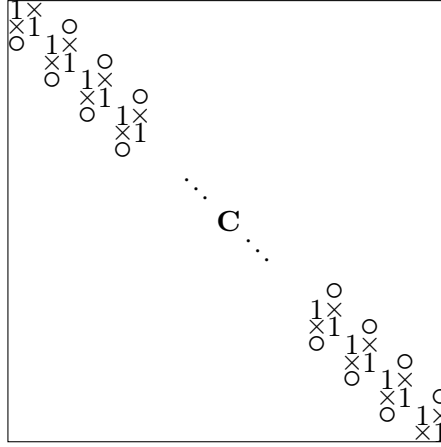


Figure 3.6: Sparsity pattern of the coupling matrix \mathbf{C} . \times -symbols are block diagonal elements different from 1, while \circ -symbols are off-block-diagonal elements.

(The \bullet -symbol indicates that the step is done *una tantum*.) It is evident that the process has been not fully parallelized, because the step 2 requires a communication between each all processes and a master process. This is why here is the point where approximations can be inserted into the SPIKE algorithm [18, pp. 502–503].

3.2 The truncated SPIKE algorithm

The sparsity pattern of \mathbf{C} is repeated in Fig. 3.6 with more detail than in Fig. 3.5, and it makes evident that the reduced system is full-coupled; nonetheless it is easy to argue a decomposition for \mathbf{C} that is akin to that used for \mathbf{A} :

$$\mathbf{C} = \mathbf{C}^{\text{BD}} + \mathbf{C}^{\text{OD}}.$$

where \mathbf{C}^{BD} is made up of the 2×2 diagonal blocks of \mathbf{C} (1s and \times -symbols in Fig. 3.6). The off-block-diagonal part, \mathbf{C}^{OD} (\circ -symbols) is made up of the top element of each column \mathbf{a}_i and the bottom element of each column \mathbf{b}_i ; ⁵ furthermore, \mathbf{a}_i and \mathbf{b}_i are (proportional, through α_i and β_i , to) the last and first columns of \mathbf{A}_i^{-1} . In summary, \mathbf{C}^{OD} is made up of the most upper-right and lower-left elements of matrices \mathbf{A}_i^{-1} .

⁵For completeness, \times -symbols are the bottom elements of each column \mathbf{a}_i and the top elements of each column \mathbf{b}_i .

Theorem 2.4 [58, pp. 493–494] guarantees that, for a positive definite and m -banded matrix \mathbf{M} , the following inequality holds

$$|\mathbf{M}^{-1}(k, l)| \leq C\lambda^{|k-l|},$$

where both C and λ depend on the condition number κ of \mathbf{M} . In particular

$$\lambda = \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^{2/m} \quad \text{which implies that} \quad \lambda < 1,$$

thus guaranteeing the elements of \mathbf{M} to decay exponentially with increasing distance from the main diagonal. Matrices \mathbf{A}_i^{-1} are just the kind of matrices mentioned in the hypothesis of the theorem — positive definite, banded matrices — so it can be used. Translating the consequence of this theorem on \mathbf{A}_i onto \mathbf{C} , the \circ -elements of the latter matrix decay exponentially with increasing M_i , and so are negligible if $M_i = \mathcal{O}(10^2)$ or more. It is clear that the approximation on which the truncated SPIKE algorithm is based is indeed to set $\mathbf{C}^{\text{OD}} = 0$ and, in turn, to make the approximation $\mathbf{C} \approx \mathbf{C}^{\text{BD}} \doteq \tilde{\mathbf{C}}$, neglecting \circ -elements; the reduced system is thus approximated by the following

$$\tilde{\mathbf{C}}\mathbf{y}'^{\text{C}} = \mathbf{q}^{\text{C}}, \quad (3.22)$$

It is now crucial to understand, with the aid of Fig. 3.4, that each 2×2 diagonal block of $\tilde{\mathbf{C}}$ does *not* pertain to a single process, as much as the corresponding two elements of the known term; on the contrary, each of these 2×2 systems is shared among two adjacent processes; as a consequence, the method is still not perfectly parallel, but at least it does not require global communications with a single process elected master. This detail makes a huge difference, as explained in Section 4.9.2.

3.3 Other approximate parallelizations

As a side work, the author participated to the development of a conservative overlap method for multi-block parallelization of compact finite-volume schemes [59]. The main idea behind the work, is the following. For each block, an enlarged linear system is solved by overlapping a certain number of neighbour cells from adjacent sub-domains. As a consequence, the two processes on each side of the block-to-block boundary compute a different value for the unknown on that boundary, thus endangering the

conservation of the method. Unlike other approaches, conservation is enforced by properly re-computing the common interface value between two neighbouring blocks by reapplying the same compact formula, and involving as known terms the maximum possible number of available data (see [59, Sec. 3.2] for details).

Chapter 4

Code walk-through

This chapter contains a thorough description of the most important components of the source code. For this reason, it should provide the reader with enough information that allow him to modify the program with awareness. The comments are related to the implemented numerics, and highlight the importance of the motives behind the numerous choices and assumptions made during the development.

4.1 Foreword

The difficulty level inherent in the work conducted by the present author emerges from the few specifications around which the main key points for the development of the code where addressed.

- A few specific test-case should be run, namely the Taylor-Green Vortices, the Lid-Driven Cavity, and the Channel flow.
- Compact schemes should be used to discretize the spatial operators, but the Harlow-Welch formulation should be allowed (so that testing against compact schemes-related bugs is feasible).
- The computational grid should be three-dimensional, structured, and Cartesian, and allow a variable spacing.
- A three-dimensional domain decomposition should be used, and the problem of the process-wise resolution of the linear systems arising from the use of compact schemes should be addressed by means of the SPIKE algorithm.

- It should be possible that the convective operator be cast in its divergence and skew-symmetric forms.

The preceding requests translate into the following, lower-level code specifications.

- The code is requested to deal with both periodic and non-periodic directions, implementing boundary conditions in the latter case.
- The use of classical explicit finite differences should be allowed. Compact schemes of several orders of accuracy should be available in a library (both explicit and compact).
- The coefficients of the schemes should be computed at runtime, and boundary conditions should adapt to several different inner schemes.
- MPI should be used to run the code on a three-dimensional grid of logical processes.
- Both divergence and advective forms for the convective terms should be allowed, since the skew-symmetric one is the average of the two.

It is crystal clear that the opportunities for hard-coding are precluded, and should not be taken either, since hard-coding, in the context of such a large project, implies a lot of copy-and-paste — which is common error-prone practice —, little re-usability of the code, few opportunities for the compiler to optimize, and several other well-established drawbacks, not to mention that the unavoidable debugging process should be conducted on source files several-fold longer than necessary.

4.2 Need for Git

Given the estimated size of the project, a version control system (VCS) was a foreseeably essential tool for carrying out a successful development, without the risks of severely messing up the code while trying to implement a feature, or forgetting the reason why an apparently cryptic edit was done. It also allows comfortably switching among multiple development branches, with no risk of confusion; those branches naturally spring up when implementing some features proves to be too hard, unless another preparatory feature is implemented beforehand.

The chosen VCS was `git`, since it is widely considered unrivalled in the software development community, due to its amenable features and reliability [60–62], and the author had to build his knowledge of `git` from the ground up, at the same time as coding.

4.3 Input/output

The program is designed to read all the input from the standard input device `/std/stdin`, print *part of* the output to the standard output device `/std/stdout`, and send errors to the standard error device `/std/stderr`. These three channels are always opened, and allow a convenient interface with the Unix shell-based user interfaces set up on almost every HPC facility. In this respect, the Bash script `naples` is used as a wrapper to the actual program, i.e., it is a layer around the true Fortran executable, and constitutes a user interface handling input/output and options in much the same way as any standard Unix tool does.

Concerning the output, a small fraction of it consists of some quantities useful to monitor the running simulation (such as time, mean velocity, energy, number of iterations for the elliptic solver to converge, wall clock time, ...), which are printed to screen at each time step. A far bigger part of output, referable to as *bulk* output, is represented by the complete flowfield, saved to binary files on a time-step or time-interval basis, as specified in the input file through the `write_step` key.

One typical command that the author used to enter, most frequently on the workstation, is the following,

```
$ naples -n <#procs> -b <out. dir.> -R < <in. file> >> <out. file> &
```

where `-n <#procs>` sets the number of MPI processes, `-b <out. dir.>` selects the directory where the flow field binary files are piled up, `-R` requests that the simulation be restarted from the latest-time flowfield contained in `<out. dir.>` (if no file is available in there, an error is thrown). The classic shell scripting redirectors `<` and `>>` are used to select `<in. file>` as the input file, and `<out. file>` as the file to which the screen output must be appended; the ampersand `&` runs the program in background, to let the shell available.

4.3.1 Example input file

An example `<in. file>` is the following, which can be regarded as a template, as well. It contains lines of space separated words, consisting of one or

more identifiers, followed by one or more values (numbers, strings, ...). Comment lines starting with # are allowed, and are used in the example to provide information about the various input lines, the meaning of which is clarified in the following sections.

```
# PHYSICS
# Initial conditions
I.C.      Poiseuille
perturbations 0.05
# Reynolds number (negative means inviscid flow)
Re        395

# OUTPUT
# interval between two successive outputs to file
# ( <= 0: don't write, integer: time steps, real: time)
#write_step      -1
write_step       100
#write_step       5e-3

# GEOMETRY
# domain lengths (negative values multiplied by  $-\pi$ )
lengths      -2      -1      2
# periodicity (1 = true)
periodic      1      1      0
# boundary conditions
uvw_BC Left   none
uvw_BC Right  none
uvw_BC Down   none
uvw_BC Up     none
uvw_BC Back   steady_solid_wall
uvw_BC Front  steady_solid_wall

# FORCES
gravity  1  0  0

# TIME
# integrator (ExplEuler, RK4)
time_integrator RK4
# stop criterion (time <num> or steady_state)
stop_at time 500
# maximum number of time steps
max_time_steps 1000000000

# NUMERICS
# convective term (D,A,S for divergence, advective, skew-symm.)
AdvDivSkew      S
# elliptic solver (J = Jacobi, SRJ = Scheduled Relaxation J.)
```

```

ell_solver          SRJ
# Courant–Friedrichs–Lewy number
CFL                 .3
# number of cells per dimension
N                   32      32      32
# spacings along each dimension (integers)
spacings            1        1        1
# selection of finite difference schemes
scheme dir1_der0_c2f A_L     4C_L     4C_C     4C_R     A_R
scheme dir2_der0_c2f A_L     4C_L     4C_C     4C_R     A_R
scheme dir3_der0_c2f A_L     4C_L     4C_C     4C_R     A_R
scheme dir1_der0_f2c A_L     4C_L_b   4C_C     4C_R_b   A_R
scheme dir2_der0_f2c A_L     4C_L_b   4C_C     4C_R_b   A_R
scheme dir3_der0_f2c A_L     4C_L_b   4C_C     4C_R_b   A_R
scheme dir1_der1_c2f          3C_L     4C_C     3C_R
scheme dir2_der1_c2f          3C_L     4C_C     3C_R
scheme dir3_der1_c2f          3C_L     4C_C     3C_R
scheme dir1_der1_f2c 3E_LL   4C_L_b   4C_C     4C_R_b   3E_RR
scheme dir2_der1_f2c 3E_LL   4C_L_b   4C_C     4C_R_b   3E_RR
scheme dir3_der1_f2c 3E_LL   4C_L_b   4C_C     4C_R_b   3E_RR
scheme dir1_der2_c2c          5C_L     6C_C     5C_R
scheme dir2_der2_c2c          5C_L     6C_C     5C_R
scheme dir3_der2_c2c          5C_L     6C_C     5C_R
scheme dir1_der2_f2f          5C_L     6C_C     5C_R
scheme dir2_der2_f2f          5C_L     6C_C     5C_R
scheme dir3_der2_f2f          5C_L     6C_C     5C_R

# TOLERANCES
# stationary reached
vel_tol             1e-3
# elliptic solver
ell_tol             1e-6

# MPI
# number of processes per direction (integers; 0 = MPI decides)
np                  0        0        0

```

4.3.2 Parallel-independent output

The output to `/dev/stdout`, or “to screen” from a user’s perspective, is left to the root process and, as such, it is clearly independent of the number of processes running the simulation. The so-called bulk output, on the contrary, is made up of all the portions of the flowfield pertaining to every process running the simulation.

The initial approach was to make each process write the flowfield rel-

ative to its subdomain to a file, and append the process id to each. This clearly ruled out the possibility of restarting a procedure on a different grid of processes, and also made the development of an external post-processing tool necessary to the visualization of the flowfield and/or the computation of derived quantities.

The second, adopted approach is to use MPI-2 input/output APIs to make all processes print the whole flowfield to a single file, concurrently. Parallel output is thus accomplished through two subroutines: one is named `blk3D_set_output` and bounded to the `velocity` variable (of class `Blk3D`, cf. Listing 4.1, Section 4.7), and it is called only at startup to define the `MPI_fileview` member, describing the view of a process over the file (i.e. the portion of the file visible to the processes), as well as the member `MPI_inner_block`, describing the non-overlap portion of the three-dimensional array to be printed; the other subroutine (`save_field`) simply uses these two datatypes to coordinate the simultaneous writes to file (through `MPI_FILE_WRITE_ALL`).

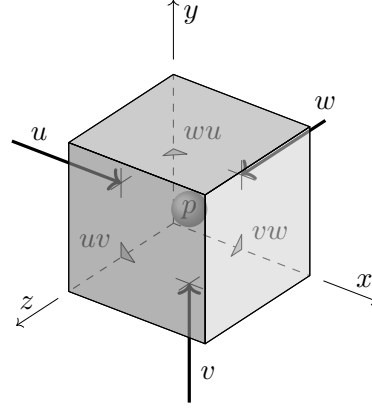
Deriving an almost perfectly specular subroutine to read in the flowfield (`load_field`, which uses `MPI_FILE_READ_ALL`) was an easy task, and allowed a comfortable use of the restarting procedure. Restarting a simulation on a different mesh (finer or coarser) is not possible within the Fortran program, since it would need interpolation operators acting between two completely different meshes. The work-around is easily found: interpolating the output of `save_field` on a different mesh is easily done through MATLAB's `interp3` function (or the like), whose output can be read in by `load_field`. In this respect, the fact that the column-major indexing is common to both languages, greatly simplifies the workflow.

4.4 Layout of variables and array indexing

The arrangement of fluid-dynamic variables is fully staggered, according to the well known layout extensively used in the incompressible CFD community [30], and it poses some code design problem, especially when non-periodic directions are concerned, as it is the case for the code in object.

Fig. 4.1 shows one elemental computational cell (the building block of the three-dimensional Cartesian domain), along with the fluid-dynamic variables relative to it: the pressure (p) is located at the cell center, and the velocity components (u , v , w) at the center of the faces, with each component resulting staggered half a cell away from the pressure, in the

Figure 4.1: Arrangement of fluid-dynamic variables around an elemental, *non-boundary*, computational cell. The 7 labelled variables (u, v, w, p, uv, vw, wu), together with the 3 pure products of velocity components (u^2, v^2, w^2 , located at cell center together with the pressure p), all belong to the same depicted cell and all share the three Cartesian indices (i.e., each of them is the element (i, j, k) of the respective variable). The velocity components through the three front faces, as well as the products of components at the 9 remaining edges, belong to the neighboring cells. Section 4.4.1 contains a thorough description of the indexing logic, with boundary conditions considered.



negative direction of the component itself.

Due to the mutual staggering of the velocity components, the quadratic quantities (e.g., the convective term) cannot be computed without interpolations or staggered differentiations of both factors. In this respect, the choice is made that each component is interpolated/differentiated along the direction pertaining to the other component [37]. As a result, the mixed products (uv, vu, wu) are located at the cell edges, whereas the pure ones (u^2, v^2, w^2) belong to the cell center (the same location as the pressure p in Fig. 4.1).

4.4.1 Indexing of the grids

For the sake of simplicity, we will refer in the following to the two-dimensional layout depicted in Fig. 4.2, since it proves very useful, while not compromising the generality. Indeed, since no third order tensors is present in the Navier-Stokes equations, no term involves the three velocity components together, so no variable needs to be defined at cell vertices; as a consequence, the two-dimensional projection of a plain of cells is representative of the mutual interaction between the variables.

In order to understand the variable indexing logic explained in the following, it is crucial to stress the meaning of the legend entries in Fig. 4.2:

- the black arrows indicate *ordinary* locations, existing along both periodic and non-periodic directions;

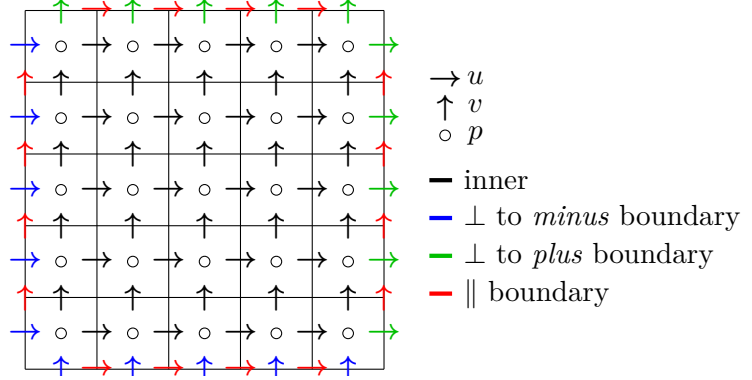


Figure 4.2: Arrangement of variables in the two-dimensional case (the whole domain, not just a process-bounded subdomain). Note that inside the domain, it is exactly the *markers and cells* (MAC) arrangement [30], whereas u (resp. v) is defined also on upper and lower (resp. left and right) boundaries $h/2$ -staggered. (In the legend, *minus* means left/down/back; likewise *plus* means right/up/front.)

- the red arrows indicate *special* locations of components tangential to the physical boundary, existing only along non-periodic directions;
- blue arrows indicate locations that exist in both cases, but are *special* along non-periodic directions, and *ordinary* along periodic directions;
- green arrows indicate *special* locations, existing only along non-periodic directions (along periodic directions they are a visual copy of the corresponding blue arrows).

In order to clarify the distinction between *ordinary* and *special* locations, it is beneficial to sketch a 10 cells long mesh, in both the periodic and non-periodic cases.

PERIODIC In Fig. 4.3: the horizontal component is located at the cell faces, and takes its index from the cell on its right; the vertical component belongs to cell centers (with respect to the considered horizontal direction), and takes its index from the cell itself.¹

¹In the more general three-dimensional case, the sentence can be reworded as *a velocity component is located at cell faces along its direction and takes its index from the cell in the minus (i.e., left, down, or back) direction, whereas, along the other two directions, it is at cell centers and takes the index from the cell itself.*

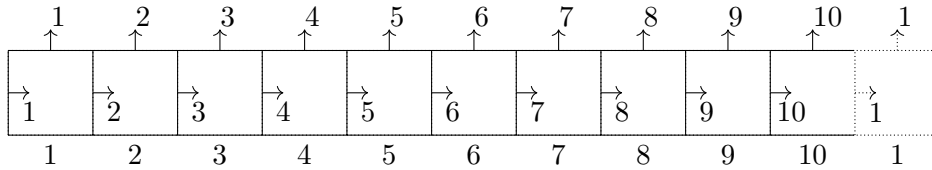


Figure 4.3: A periodic mesh consisting of ten cells. Indexed elements from the bottom up: cells, horizontal components, vertical components. The 11th cell and components do not exist, since they coincide with the 1st ones (just like the green arrows do not exist in Fig. 4.2 since they coincide with the blue ones). (Note that this is the whole mesh, not just a process-bounded sub-mesh.)

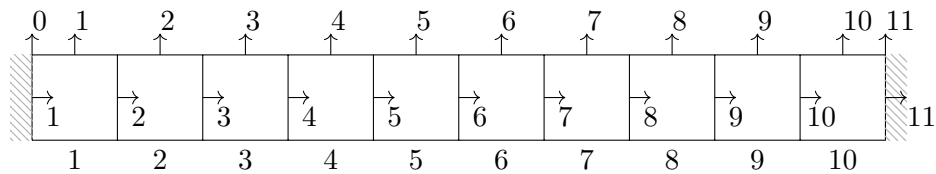


Figure 4.4: A non-periodic mesh consisting of ten cells. Indexed elements from the bottom up: cells, horizontal components, vertical components. On the two left and right physical boundaries of the domain both components are defined, with the location 11 being *ordinary* for the horizontal component, whereas the locations 0 and 11 are *special* for the vertical component. (Note that this is the whole mesh, not just a process-bounded sub-mesh.)

Table 4.1: First index, last index, and length of the grids along the non-periodic x direction. The values corresponding to $n_x = 10$ can be verified against Figs. 4.3 and 4.4.

quantity	bounds		length
	lower	upper	
p	1	n_x	n_x
u	1	$n_x + 1$	$n_x + 1$
v	0	$n_x + 1$	$n_x + 2$
w	0	$n_x + 1$	$n_x + 2$

NON-PERIODIC The numbering is almost the same, but it is crucial that some other indices be introduced, as sketched in Fig. 4.4.

- An 11th face is introduced on the right physical boundary, as opposed the 1st face on the left physical boundary. On this newly introduced *special* location, the normal velocity component is defined, and assigned through the boundary condition.
- The vertical velocity component, which is ordinarily defined at cell centers along the horizontal direction, must be assignable on the left physical face — through the tangential boundary condition —, which is not its natural position. The index for this *special* location is set to 0, so as to avoid shifting the indices of all the following points, which would introduce a useless and inconvenient difference in the indexing with respect to the periodic case.
- Similarly, a boundary condition for the tangential velocity on the right boundary must be allowed as well, so another *special* location for the cell centered vertical component is the right physical boundary, where the index 11 is assigned to the velocity. It is worth to explicitly specify that the fact that the point 11 is the 12th point, if starting counting from 1, has no importance at all.

Lower and upper bound of the grids along the direction x discretized with n_x cells are reported in Table 4.1, along with the extent.

Fortran can start indexing from non-positive integers

The design choice outlined so far fits perfectly with the Fortran programming language, which naturally allows arrays be indexed starting from any integer, beside the default, 1. Indeed, an explicit-shape array can be declared in Fortran by either of the following two lines

```
<type>, DIMENSION (10) :: x1
<type>, DIMENSION (-4:5) :: x2
```

the difference being in the indices of the first and last elements, which can be accessed through `x1(1)` and `x1(10)`, and `x2(-4)` and `x2(5)` respectively; those bounds can be extracted through the intrinsic functions `LBOUND` and `UBOUND`. A complication consequent to the use of lower indices other than

1, is that explicit-shape arrays got passed to a procedure (**FUNCTION** or **SUBROUTINE**) along with their size (accessible through **SIZE**), not their lower and upper bounds. Indeed, the assumed-shape dummy arguments in the following subroutine, corresponding to the `x1` and `x2` actual arguments,

```
CALL mysub(x1,x2)
...
SUBROUTINE mysub(x1_dummy,x2_dummy)
  <type>, DIMENSION(:) :: x1_dummy
  <type>, DIMENSION(:) :: x2_dummy
  PRINT *, LBOUND(x1_dummy) ! prints 10
  PRINT *, LBOUND(x2_dummy) ! prints 10
```

both have the same bounds 1 and 10, so `x2`'s lower bound, which I claimed useful if not invaluable, gets lost. The selected solution, among the available ones, was to give up on static memory allocation: on the one hand, explicit-shape arrays are abandoned in favor of allocatable arrays — the flagship of dynamic memory allocation —; on the other, dummy arguments are declared as deferred-shape arrays (available as of Fortran 2003). As a result, the array gets passed along with its bounds.

```
<type>, DIMENSION(:), ALLOCATABLE :: x
...
ALLOCATE(x(-4:5))
CALL mysub(x)
...
SUBROUTINE mysub(x_dummy)
  <type>, DIMENSION(:), ALLOCATABLE :: x_dummy
  PRINT *, LBOUND(x_dummy) ! prints -4
...
```

In addition to allowing the comfortable use of arrays with lower bounds other than 1, the use of allocatable arrays, in place of standard ones, brings several advantages and resolves disadvantages of old Fortran programming, as thoroughly expressed throughout good Fortran manuals [63].

4.5 The Cartesian grid of MPI processes

When the program is run in parallel on p processes, an MPI communicator with a three-dimensional Cartesian topology attached to it (cf. Fig. 4.5a), namely `procs_grid`, is created through `MPI_CART_CREATE`.²

²As explained in Section 6.3.1, the program cannot run with less than 2 processes per direction, so it is the user's responsibility to request, through the option `-n(<#proc.s>)`,

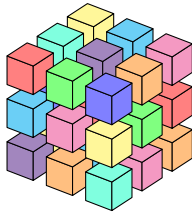
Afterward, the subroutines `MPI_CART_COORDS`, `MPI_CART_SHIFT`, and `MPI_CART_SUB` are used to make each process aware of its “role” in the topology, and each bit of information is used in several parts of the code.

- The identities of the two neighbors in the generic direction $\langle dir. \rangle$, are stored in `idm($\langle dir. \rangle$)` and `idp($\langle dir. \rangle$)`, with `m` and `p` standing for *minus* and *plus* respectively. `idm` and `idp` are mainly used for two reasons:
 - as source and destination ranks in send-like and receive-like MPI APIs;
 - to determine if the current process is adjacent to the boundary, through the checks `idm($\langle dir. \rangle$) == MPI_PROC_NULL` and `idp($\langle dir. \rangle$) == MPI_PROC_NULL` (`MPI_PROC_NULL` means “none”).
- The handles to the three orthogonal *pencils*, which the process belongs to (one-dimensional communicators, e.g., the yellow and green processes in Fig. 4.5b), are stored in the array `pencil_comm(1:3)`, and are used for
 - the communications relative to the SPIKE algorithm, which acts on one-dimensional operators,
 - the communication of pure quadratic convective terms (u^2 , v^2 , w^2 , uu_x , vv_y , ww_z), which need be exchanged only along one direction.
- Similarly, the handles to the three orthogonal *slabs*, which the process belongs to (two-dimensional communicators, e.g., the blue and green processes in Fig. 4.5b), are stored in the array `slab_comm(1:3)`, and are used only the communication of mixed quadratic convective terms (uv , vw , wu , uv_x , vw_y , wu_z , u_yv , v_zw , w_xu), which need be exchanged along two directions.

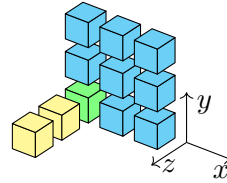
4.6 Process-local array indexing

Once the Cartesian communicator `procs_grid` is created — which means that the number of processes along each direction is known, stored in the

a total number of processes p which is the product of at least three integers greater than 1; provided this condition, `MPI_DIMS_CREATE` determines a balanced dimension $P_1 \times P_2 \times P_3$ for the grid, such that $p = P_1 P_2 P_3$.



(a) Sketch of the three-dimensional MPI topology with a $3 \times 3 \times 3$ process grid. This is the three-dimensional communicator `procs_grid`, whose dimensions are stored in `dims(1:3)`.



(b) A process (green) with two of the seven communicators it belongs to, e.g., `pencil_comm(3)` and `slab_comm(3)`. All communicators contain the current process as well.

Figure 4.5: Communicators with Cartesian topology attached. Processes in (b) are extracted from (a).

global variable `dims(1:3)` —, the total numbers of cells along the three directions, `Ntot(1:3)`, can be distributed among the processes.

Essentially, the integer division of the total number of cells by the number of processes is computed; then it is augmented by 1 for the leading r processes, where r is the remainder of the integer division. At the same time, the cells local to each process are indexed starting from 1; then the indexing of cell-centered and face-centered locations is inherited from that of the cells, according to Section 4.4, and extended to take overlaps into account. The resulting indexing pattern is shown in Fig. 4.6, where a non-periodic mesh is distributed among three processes. Note that the middle process, which handles 5 cells, allows overlaps on both sides, and is therefore illustrative of the periodic case (in which “all processes are in the middle”). Similarly, the other two processes use the same overlap indexing, and share Fig. 4.4’s *special* indexing of boundary variables.

4.7 The Blk3D class

Every scalar local fluid-dynamic variable is stored in a rank-3 array resembling the three-dimensional geometry of the physical domain. Given the conventions about the one-dimensional indexing explained in Section 4.4.1 and adopted in Figs. 4.3, 4.4 and 4.6, it is clear that each fluid-dynamic variable has its own indexing based on where it is located (at center/-face/edge/... of the cell), even though those indices are tightly related to each other. Instead of hard-coding some offset variables, manually used to

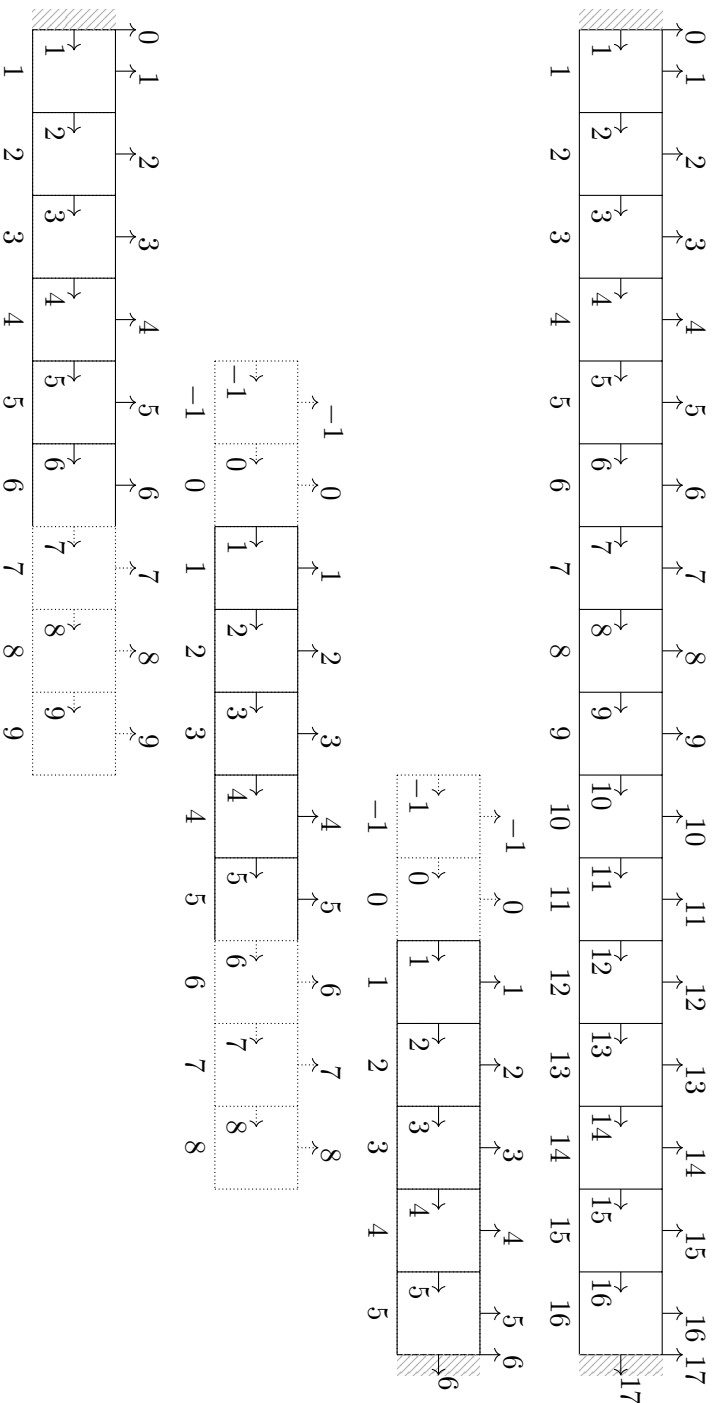


Figure 4.6: Process-local mesh indexing, as compared to global indexing, for a non-periodic mesh distributed among three processes. The assumption is made that the variable in question has two overlap layers in the minus direction and three in the plus direction. The central process is representative of the periodic case as well, where no physical boundaries exist, and the overlap is on both sides. Note that there is no ambiguity in the fact that zero is used to index the first overlap on the minus side and the boundary for cell-centered variables, since the two possibilities cannot coexist.

treat differently the various variable-containing arrays, the choice was to create the derived datatype `Blk3D` — typeset in Listing 4.1 — which contains the aforementioned rank-3 array (in the member `values`), together with a set of other useful members bounded to it, indices included.

The fundamental variables of class `Blk3D` are declared as follows,

```

TYPE (Blk3D), DIMENSION (ndims), TARGET :: velocity
TYPE (blk3D), DIMENSION (ndims, ndims)    :: velvel, dervel
TYPE (blk3D), DIMENSION (ndims)          :: convvel, diffvel

```

(where `ndims` is a **PARAMETER** with value 3, since the solver is intrinsically three dimensional) and their content is as follows, using the same notation adopted by Morinishi et al. [37],

- `velocity(i)` unsurprisingly contains u_i ;
- `velvel(i, j)` contains $\bar{u}_j^{x_i}$ at the beginning of a time-step, and $\bar{u}_j^{x_i} \bar{u}_i^{x_j}$ at the end of it.
- `dervel(i, j)` contains $\delta u_j / \delta x_i$ at the beginning of a time-step, and $\bar{u}_i^{x_j} \delta u_j / \delta x_i$ at the end of it.
- `convvel(j)` contains $\sum_i \bar{u}_i^{x_j} \delta u_j / \delta x_i$.
- `diffvel(j)` contains the approximation of $\sum_i \partial^2 u_j / \partial x_i^2$.

With reference to the generic three-dimensional scalar field stored in the variable `<var.>` of type `Blk3D` among those just listed, all these informations are contained in `<var.>%b`, `<var.>%b_bc`, `<var.>%b_bo`, and `<var.>%b_ol`, as follows.

- The integer `<var.>%b(<dir.>, 1:2)` stores the indices of the first (1) and last (2) elements along the `<dir.>` direction, *excluding* boundaries and overlaps; it is determined as soon as the cells are distributed among processes.
- `<var.>%b_bc(<dir.>, 1:2)` stores the same indices when boundaries are *included* and overlaps are *excluded*; also this array is determined just after the cell are distributed.
- `<var.>%b_ol(<dir.>, 1:2)` stores the same indices when boundaries are *excluded* and overlaps are *included*; this array is stored *after* the

compact schemes are build, since the number of overlap layers depends on the number of sub- and super-diagonals of the **B** matrix, i.e., on the members `ld` and `ud` of the `CDS` type.

- `<var.>%b_bo(<dir.>,1:2)` contains a copy of the corresponding element of the `%b_bc` member if the process is adjacent to the boundary along that oriented direction, otherwise a copy of the same element from `%b_ol`; it is computed just after `<var.>%b_ol(<dir.>,1:2)`.

Concerning the MPI exchanges, since they are performed in order to make available all the quantities to each process, so that each can compute derivatives and interpolants, it is clear that the exchange should occur only along certain directions; for instance, each velocity components must be exchanged in all three directions, since each of them undergoes computations along all directions, whereas *uv* undergoes differentiations only along the directions *x* and *y*, so the exchange along *z* should not take place. In order to take these informations into account, the **POINTER** member `<var.>%comm` points to the appropriate MPI Cartesian (sub-)communicator's handle (either the pencil or slab sub-communicators in Fig. 4.5b, or the whole three-dimensional grid of Fig. 4.5a).

Listing 4.1: `Blk3D` class.

```

MODULE class_Blk3D
...
TYPE, PUBLIC :: Blk3D
  ! Rank-3 array containing the 3D flowfield
  REAL,      DIMENSION(::,::,:), ALLOCATABLE :: values
  ! Lower and upper indices
  INTEGER, DIMENSION(ndims,2) :: b
  INTEGER, DIMENSION(ndims,2) :: b_bc
  INTEGER, DIMENSION(ndims,2) :: b_ol
  INTEGER, DIMENSION(ndims,2) :: b_bo
  ! Communicator and datatype to exchange
  INTEGER, POINTER :: comm
  INTEGER, DIMENSION(:), ALLOCATABLE :: &
    & array_of_s_types, array_of_r_types
  INTEGER, DIMENSION(:), ALLOCATABLE :: &
    & array_of_s_counts, array_of_r_counts
  INTEGER(MPI_ADDRESS_KIND), DIMENSION(:), ALLOCATABLE :: &
    & array_of_s_displs, array_of_r_displs
  ! Data-types to write to file
  INTEGER :: MPI_inner_block, MPI_fileview

```

```

CONTAINS
PRIVATE
  ! Specific type-bound procedures for generic support:
  PROCEDURE, PASS :: blk3D_set_output
  PROCEDURE, PASS :: blk3D_exchange
  ...
PUBLIC
  ! Ordinary type-bound procedures
  ! Generic type-bound procedures, assignment (=) and operators (+, -, *, /, ...)
  ...
END TYPE Blk3D
CONTAINS
  ! Body of the procedures
  ...
END MODULE class_Blk3D

```

4.8 Compact operators

One compact schemes is defined for each of the several operators acting on the fluid-dynamic variables.

- Firstly, there are operators to compute interpolants, as well as first and second derivatives;
- furthermore, for each order of differentiation (0, 1, 2) two operators are required, one for center-to-face and center-to-center operations (interpolation and first differentiation, and second differentiation, respectively), the other for face-to-center and face-to-face operations (ditto);
- finally, these schemes must be allowed to be different along the three direction, at least to allow both periodic and non-periodic directions (the former do not need boundary schemes, unlike the latter).

With these three requirement outlined, all the needed compact schemes are grouped together in the rank-3 array variable declared as

```

TYPE(compact_type), DIMENSION(ndims,0:2,2), TARGET :: cmp

```

each element of which is of type `compact_type`, a user defined derived data type to be defined shortly. The general element `cmp(i, j, k)` contains the compact operator for the j^{th} derivative, along the direction i , operating on values located at cell centers if $k = 1$, or at cell faces if $k = 2$.

Since compact schemes consist of two matrices, one multiplying the array of unknowns, the other multiplying the array of knowns, and both these matrices are banded, it is natural that `compact_type` contains the LHS and RHS matrices as members.³ Because the specific schemes are selected through some identifying strings expected in input (see Section 4.8.1), it is natural, as well, to include another member aimed at storing those strings, as soon as they are read from the input file, before parsing them at the time that the matrices are populated with coefficients. The definition of `compact_type` is thus the following,

```

TYPE compact_type
  TYPE (CDS) :: A
  TYPE (CDS) :: B
  CHARACTER(LEN = 20), DIMENSION(:), ALLOCATABLE :: sch
  INTEGER :: N
END TYPE compact_type

```

- The `sch` member is an array of strings, 20 characters long at most, whose central element `sch(0)` is aimed at containing the identifier of the scheme used in the inner part of the domain (generally a symmetric scheme), while the surrounding elements `sch(<#left sch>:-1)` and `sch(1:<#right sch>)` must contain the boundary and near-boundary schemes. Clearly, in the periodic case only the central element is used, while surrounding strings are ignored, if provided.⁴
- `N` is a member of convenience to easily access the number of equations relative to a compact scheme, a number otherwise retrievable from either `A` or `B` member, which are the core of `compact_type`.
- The members `A` and `B` are aimed at containing the true differential operators, in the form of banded matrices filled with the coefficients determined at runtime, based on the selected schemes (saved in the `sch` member) and on the metrics of the mesh. Both variables are of type `CDS`, a user defined derived data type described in Section 4.8.2 and used to store banded matrices in Compressed Diagonal Storage format, as defined in [64] (Fig. 4.7 is self-explanatory in this regard).

³The term “member” is the name used for variables/functions constituting C++’s **structs**, but it is preferred here, since the standard Fortran’s term “component”, could cause confusion with the components of a vector or array.

⁴This automatic discarding of unnecessary boundary schemes is useful, since it allows switching on/off the periodicity of a direction with no need to change the scheme-specification part of the input file.

4.8.1 Selection of finite difference schemes from input

Before going in depth into the `CDS` derived data type, it is worthwhile to describe how the finite difference schemes are selected, from those available in the library, on the basis of the corresponding input lines, i.e., what is the meaning of the `sch` member. A typical line consists of an odd number of space separated strings, e.g.,

```
scheme dir3_der0_f2c A_L 4C_L_bidiag 4C_C 4C_R_bidiag A_R
```

which undergoes the following processing.

1. The word `scheme` is the keyword to tell the parsing routine that the sourced line selects numerical schemes, so it is thrown away once read;
2. the underscore-separated string `dir3_der0_f2c` specifies that all the following strings, up to the EOL, must be used to select schemes for the interpolation (`der0`) from faces to centers (`f2c`) along the 3rd direction (`dir3`); once read, these three directives are used to select the operator `cmp(3,0,2)` (`c2c` and `c2f` are converted to 1, whereas `f2c` and `f2f` are converted to 2);
3. each of the following five strings (n in general, with n odd by necessity) `A_L`, `4C_L_bidiag`, `4C_C`, `4C_R_bidiag`, and `A_R`, is conveniently appended to `der0_f2c` and the result is collected into `cmp(i,j,k)%sch(-2:+2)`, symmetrically around the element of index zero.
4. (a) If the direction i is non-periodic, the two strings `A_L` and `4C_L_bidiag` stored in `cmp(i,j,k)%sch(-2:-1)` are used to determine the coefficients with which the first two rows of the matrices `cmp(i,j,k)%A%matrix` and `cmp(i,j,k)%B%matrix`, are filled (starting the count from the topmost one); likewise, the last two rows are filled based on `cmp(i,j,k)%sch(+1:+2)` (strings `4C_R_bidiag`, and `A_R`); all other rows are determined according to `cmp(i,j,k)%sch(0)` (string `4C_C`).
- (b) If the direction is periodic, all the rows are equally filled with coefficients of the scheme specified by `cmp(i,j,k)%sch(0)`.

It is worthwhile to note that the strings `A_L`, `4C_L_bidiag`, `4C_C`, `4C_R_bidiag`, and `A_R` of the input line, once the string `der0_f2c` is

prepended to each of them, uniquely identify a scheme in the library, since they are used as case selectors in a `SELECT CASE` construct. Upon inserting a new scheme in the library, the identifying string can be chosen with complete freedom, as long as it does not coincide with an already existing identifier. For instance, the string `der0_f2c_A_L` was chosen to identify an Assignment equation for the Left boundary variable, whereas the string `der0_f2c_4C_C` was chosen to identify the fourth order central compact scheme for the interpolation. The trailing `_bidiag` substring in `der0_f2c_4C_R_bidiag` was put there just to distinguish a newly introduced scheme from an existing one, which already used the identity `der0_f2c_4C_R`.

This way of selecting the numerical schemes from input was decided by the author in order to guarantee that compact schemes of different order of accuracy could be chosen for the inner part of the domain, and that an appropriate number of asymmetric schemes could be selected on both boundaries to close the system of equation. Despite this pragmatic motive, this implementation turned out to be a very powerful feature, which provides extreme versatility in that it allows a fine-grained choice of the schemes.⁵ Indeed, despite the list of schemes is commonly of the type

$$s_{-B}, s_{-B+1}, \dots, s_{-2}, s_{-1}, s_0, s_1, s_2, \dots, s_{B-1}, s_B$$

which is used to select a main, central scheme s_0 to be used throughout the domain, except at B few points at the left and right boundary and close to them, it is also possible to assign different schemes at different mesh points, using a list like the following

$$s_1, s_2, \dots, s_N$$

or to insert one scheme \tilde{s} to be tested in the middle of a well-tested list of schemes, e.g.,

$$s_{-B}, s_{-B+1}, \dots, s_{-1}, s_0, \dots, s_0, \tilde{s}, s_0, \dots, s_0, s_1, \dots, s_{B-1}, s_B$$

The limitation about the odd number of strings, i.e., that the schemes on the left and on the right of the central one must be the same in number, is only apparent. For instance, if three boundary schemes must be applied to the right boundary and only one to the left, then it is enough to replicate

⁵Actually, at the time of writing, this is possible only along non-periodic directions.

thrice the string relative to the central scheme, obtaining the list

$$s_{-1}, \underbrace{s_0, s_0, s_0}_{\text{"balance" } s_2 \text{ and } s_3}, s_1, s_2, s_3.$$

About **c2c/c2f/f2c/f2f**

It should be already clear that the string `4c_c` actually identifies different schemes depending on the `der*_2*` string prepended to it: together with `der0_f2c` it identifies the fourth order central compact scheme for the interpolation, whereas it identifies the Padé scheme if the string `der1_f2c` is prepended.

On the other hand, leaving aside non-uniform mesh-related concerns, the Padé scheme, or any other finite difference scheme, should not depend on whether the unknown and known values are at faces and centers respectively, or vice-versa. Indeed, they do not. The importance of the substring defining the one or the other “staggering” (`c2c/c2f` versus `f2f/f2c`) lies in the ordering of variables as described in Section 4.4, which is tightly related to the ordering of equations.

For instance, the string `c2f` signifies that the variable with index j appearing at the RHS is located at a center, i.e., *after* (e.g., on the left of) the variable j appearing at the LHS, which is located at the cell face.

This distinction is essential to the correct insertion of the coefficients in the finite difference matrix operators.

4.8.2 Storage of banded matrices

The concept of the CDS format for a banded matrix, revealed in advance in Section 4.8, is soon understood upon observing Fig. 4.7. The corresponding CDS derived data type is defined as follows,

```

TYPE CDS
  REAL,    DIMENSION(:,:), ALLOCATABLE :: matrix
  INTEGER, DIMENSION(2)      :: lb, ub
  INTEGER      :: ld, ud
  INTEGER      :: lid, uid
END TYPE CDS

```

Clearly, the same type `CDS` is used for both the **A** and **B** matrices in Eq. (2.4), both of which are banded, multiply an array, and return an array. The `matrix` member is a rank-2 array, whose columns store the diagonals of

the actual banded matrix, and must be populated with the coefficients of the selected compact schemes.⁶ The lower and upper bandwidths of such matrices are computed based on the selected schemes, and their values are stored in the members `ld` and `ud` respectively, both positive integers.⁷

The meaning of the remaining members (the rank-1, dimension-2 arrays `lb` and `ub`) is related to the indexing of variables as defined in Sections 4.4 and 4.6, and is a bit tricky, at the point that a foreword is imperative, and a sketch like the one in Fig. 4.7 is of great help. With reference to Fig. 4.6 and to the compact scheme discretizing the staggered first derivative of u along the non-periodic x direction, the matrix \mathbf{B} is right-multiplied by the discretization of u , which is located at faces and indexed from 1 to 17, so these are the chosen indices of \mathbf{B} 's first and last columns; likewise, since \mathbf{A} is right-multiplied by the sought quantity $\partial u / \partial x$, located at cell centers as well as first and last physical faces, and indexed from 0 to 17, these are the indices chosen for the first and last columns of \mathbf{A} . At this point, the indices of \mathbf{A} 's and \mathbf{B} 's rows seem to have little significance, since they are nothing more than equation indices and could be freely chosen; on the contrary, it is extremely convenient to set these indices equal to those of \mathbf{A} 's columns, which eases a lot the coding of matrix-columns multiplication routines, among other advantages. To conclude the example, the indices of matrices and arrays relative to the application of the aforementioned compact scheme are collected here:

$$\begin{array}{ccccc} [0, 17] \times [0, 17] & & [0, 17] \times 1 & = & [0, 17] \times [1, 17] & & [1, 17] \times 1 \\ \mathbf{A} & \times & \mathbf{y}' & = & \mathbf{B} & \times & \mathbf{y}. \end{array}$$

as well as in Fig. 4.7. The lower and upper row indices are stored in the members `lb(1)` and `ub(1)`, whereas those of the columns are stored in `lb(2)` and `ub(2)`.

The given definitions of the members `lb` and `ub`, and `ld` and `ud`, should make clear that, once these values are computed, the member `matrix` of the generic CDS variable `<mat>` is allocated by the following statement

```
ALLOCATE (<mat>%matrix(<mat>%lb(1) : <mat>%ub(1), -<mat>%ld : +<mat>%ud))
```

⁶Actually this choice is not the most efficient, as discussed in Section 6.3.3, especially for processes adjacent to the boundary.

⁷The members `uid` and `lid` contain “alternative” values of the bandwidths, to the benefit of the performance in the case of non-periodic directions as explained in Section 4.8.3.

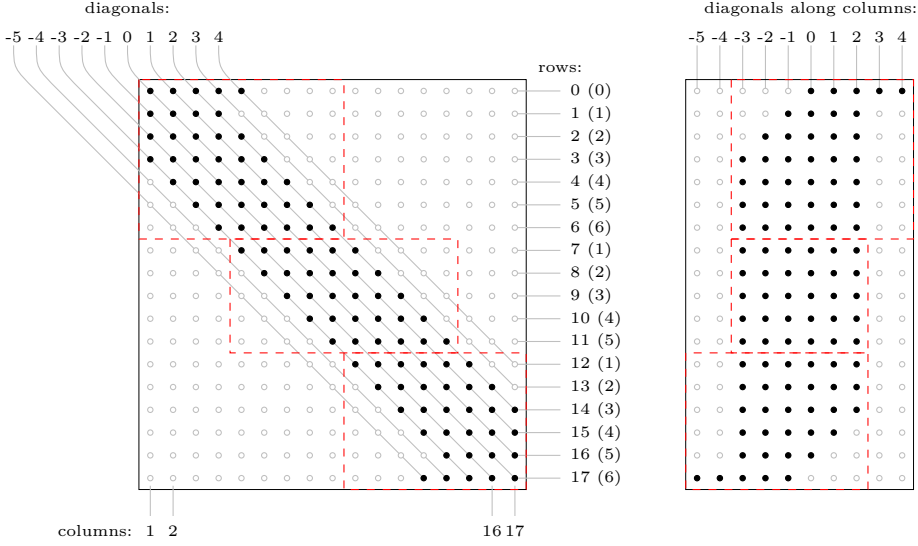


Figure 4.7: Left: sparsity pattern of a possible non-circulant \mathbf{B} matrix for the computation of the first derivative at the cell-centers and physical faces of Fig. 4.6, based on the values given at cell faces. Right: sparsity pattern of its CDS format. Indices of rows, columns, and diagonals are labelled as well, according to Fig. 4.6. The red, dashed rectangles illustrate the partitioning of the matrix among three processes (the row indices in parenthesis are indeed the *local* indices of the row/equation/unknown). See Section 4.8.3 for details.

4.8.3 Distribution of banded matrices

Once the banded matrices of compact schemes are formed by the master process, they are distributed among other processes through the APIs `MPI_ISEND` and `MPI_RECV`.⁸ The rows of \mathbf{A} and \mathbf{B} that each process must receive are those with the same indices as the locations (faces or centers) where the unknowns local to the process are prescribed. The red, dashed lines in Fig. 4.7 represent the sub-matrices relative to three processes (see Fig. 4.6 for the grids).

The number of null elements in the CDS format of the global matrix is clearly relevant, due to the boundary conditions protruding far beyond

⁸In this respect, a call to the collective `MPI_SCATTERV` could condensate the pair `MPI_ISEND/MPI_RECV`, thus allowing for further code optimization from the compiler, but it is not actually needed, since this distribution is done once and for all, and has an overall negligible computational cost.

the band relative to inner schemes. After the matrix is cut in chunks and scattered to the three processes, the problem is overcome for processes not adjacent to the boundaries, and slightly alleviated for processes touching either boundary (the zeros are almost halved in number, in the latter case). A discussion about a definitive alternative is presented in Section 6.3.3.

With reference to the `cds` datatype, the members `ld` and `ud` would store the numbers -3 and 2 respectively, in the case of Fig. 4.7.

4.9 The count of communications

The Taylor-Green Vortex test-case is used in Chapter 5 to assess the performance of the developed solver, in terms of scalability. The results are still unsatisfactory, but no more than the CFD solver in object is a brand-new code, developed in relatively little time, considering the given specifics (cf. Section 4.1).

The topic of poor performance — likely caused by a number of memory usage-related issues —, and possible strategies to remedy it, are discussed in Chapter 6. On the other hand, it is important here to compare the number of the MPI communication calls and the amount of data exchanged, as compared to 1D and 2D domain decomposition-based solvers, e.g., Incompact3D [9–11].⁹

4.9.1 Count for slab- and pencil-based solvers

When the one-dimensional domain decomposition approach is chosen, the domain is cut along one direction, as depicted in Fig. 4.8a, and each process is given one of these slab-shaped subdomains. Since each subdomain extends all the way across the domain along two directions (e.g., x and y), the process holding those data can perform all differentiations/interpolations along that two directions ($u_x, u_y, v_x, v_y, w_x, w_y, \dots$); afterward, the MPI program switches to the other state of the decomposition, and all remaining operations along z are performed. It is worthwhile to specify that the maximum number of processes allowed with this parallelization on a $N_x \times N_y \times N_z$ global grid (with $N_x \leq N_y \leq N_z$) is N_y .

Similarly, when the two-dimensional domain decomposition approach is chosen, the domain is cut along two directions, as depicted in Fig. 4.8b,

⁹The count comparison is not fair since Incompact3D uses a *partially* stagger mesh, so the products of velocity are computed with no need for “preparatory” interpolations.

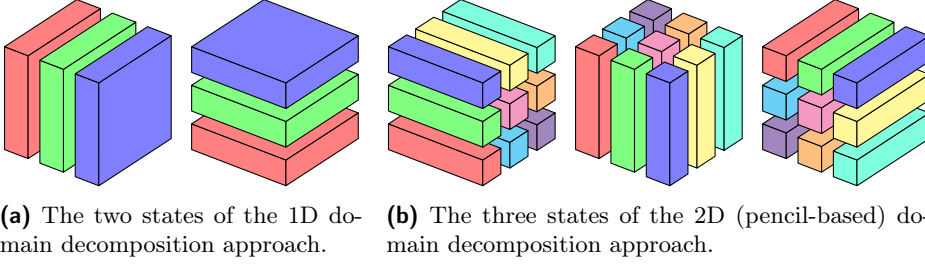


Figure 4.8: Configurations needed for slab- and pencil-transposition-based domain decompositions.

and each process is given one of these pencil-shaped subdomains, hence the name of *pencil* decomposition. Since each subdomain extends over the entire length of the domain along one direction (e.g., x), the process holding those data can perform all differentiations/interpolations along that direction (u_x, v_x, w_x, \dots); afterward, the MPI program switches to the y state of the decomposition, so that all operations along y can be performed; and so on. The maximum number of processes allowed in this cases is $N_x N_y$.

In both cases, the transposition implies a “volumetric” exchange of data among the processes, which is accomplished through the subroutine `MPI_ALLTOALLV`, at the end of which, the geometric portion of the domain held by the processes is changed (switching among two or three configurations in the two variants). This is the reason why the author refers to this decompositions by the adjective *dynamic*. In other words, all the total volume of data is moved in memory, for each variable. The total count of global transpose operations per time-step is claimed to be between 55 and 67, depending on the boundary conditions [9].

Since the *pencil*-based approach is considered superior to the other, earlier alternative, the comparison is made with it only. Furthermore, since `Incompact3D` solves the Poisson equation in the spectral space, whereas the code in question uses an inadequate iterative solver in the physical space (see Section 4.10 for details), the comparison is made after deducting the pressure-related exchanges, which add up to 4 to 16 global operations.

In conclusion, this parallelization technique requires about 51 global operations per time-step, each involving $N_x N_y$ processes, and transferring the whole volume of data N^3 , for a total of $51N^3$ floating points transferred per time-step, being N an appropriate average of N_x, N_y , and N_z .

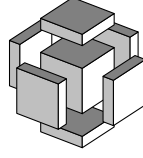


Figure 4.9: Exchange blocks relative to a subdomain, each consisting of a number of layers which depends on the specific compact schemes used. Two, four, or all blocks are exchanged, depending on the specific variable, as explained in Section 4.7.

4.9.2 Count for the developed code

The SPIKE algorithm developed in Chapter 3, allows the implementation of a *static* three-dimensional domain decomposition, within which no process holds all data along any direction, and the portion that it holds is the same throughout the entire run. In the following, the assumption is made that the meshgrid and the process grid have size $N \times N \times N$ and $P \times P \times P$, respectively, with no loss of generality. Concerning the load per process, in light of the resolution of the reduced system inherent in the SPIKE algorithm, it is necessary that the ratio $\frac{N}{P}$ be greater than some threshold τ ; as a consequence, the maximum number of processes allowed is $(\frac{N}{\tau})^3$.

For each compact scheme of the form $\mathbf{A}\psi = \mathbf{B}\varphi$, each process needs to exchange a few layers of data with the 2 neighbors in the given direction, in order to compute the product $\mathbf{q} = \mathbf{B}\varphi$; then, the “provisional” solution is computed with no exchange; successively, two pencil-local exchanges are needed to provide each process with the corrective coefficients; finally, the correction is applied with no need for further communications. Fig. 4.9 gives a visual representation of the blocks of data (each consists of few layers, depending on the specific schemes) to be exchanged for a quantity such as u (which undergoes differentiations/interpolations along all three directions); clearly only four or two blocks are exchanged for uv and u^2 , and the like (see Section 4.7). More in detail, the flow of computations and communications per time-step is as follows,

1. ℓ layers are exchanged with the 2 neighbors along the three directions for each of the three velocity component, for a total of $3^2\ell$ layers exchanged; in the present implementation, this communication is accomplished through only 3 calls to `MPI_NEIGHBOR_ALLTOALLW`, each

involving the 6 neighbors. (These calls would be condensed into 1, if the upgrade outlined in Section 6.3.4 is to be applied.) Downstream of the communication, the product $\mathbf{q} = \mathbf{B}\boldsymbol{\varphi}$ can be computed locally.¹⁰

2. Each component of velocity is interpolated and differentiated to the degrees 1 and 2 along each direction, resulting in the parallel resolution of 3^3 systems $\mathbf{A}\boldsymbol{\psi} = \mathbf{q}$, which is possible thanks to the SPIKE algorithm. In this context, each application of the algorithm requires $2P$ layers to be exchanged *twice* with the master process, through `MPI_GATHER`, before the reduced system is solved, and `MPI_SCATTER`, after. The total amount of exchanges is therefore $3^3 2^2 P$ layers, through $3^3 2$ collectives, each called within a pencil of processes. After the communications, the computation of the various interpolants and derivatives can be completed locally, and the diffusive contribution needn't undergo further exchanges.
3. ℓ layers of the 3^2 “advective” products $\bar{u}^i u_j$ are exchanged with two neighbors (each product is exchanged in only 1 direction); the same is done for the 3 pure “divergence” products $\bar{u}^i \bar{u}^i$; ℓ layers of the 3 independent mixed “divergence” products ($\bar{u}^i \bar{u}^j$ with $(i - j) \bmod 3 \neq 2$) are exchanged along 2 directions. The total is $3^2 2\ell$ layers exchanged, accomplished through 15 ($= 3^2 + 3 \cdot 2$) calls to `MPI_NEIGHBOR_ALLTOALLW`, 12 ($= 3^2 + 3$) involving 2 processes, the remaining 3 involving 4 of them. Virtually these 15 calls could be regrouped in fewer communications, just like in step 1.¹¹ Once the communications are completed, the product $\mathbf{q} = \mathbf{B}\boldsymbol{\varphi}$ can be computed locally.
4. Each communication in step 3 corresponds to the resolution of a systems $\mathbf{A}\boldsymbol{\psi} = \mathbf{q}$, so the last and final step consists of $3^2 2^2$ calls to `MPI_GATHER/MPI_SCATTER` (each called within a pencil of processes) for a total amount of $3^2 2^3 P$ layers exchanged.

The communication count conducted so far is concisely expressed in Table 4.2, where it is apparent that Incompact3D uses less than half the

¹⁰ ℓ refers to the number of layers exchanged with the processes in the *minus* and *plus* directions, added together, so it equals exactly the *inner* bandwidth of \mathbf{B} (see Section 4.8.2). Each layer consists of $\left(\frac{N}{P}\right)^2$ **REAL** numbers.

¹¹Reducing all 15 calls to 1 is questionably advantageous, due to the distance in memory of the involved arrays, but the reduction by one third should pay off

Table 4.2: Communication count.

routine	# calls	exchanged REALs per process
MPI_NEIGHBOR_ALLTOALLW	$3 + (3^2 + 3 \cdot 2) = 18$	$3^3 \ell \left(\frac{N}{P}\right)^2$
MPI_GATHER/MPI_SCATTER	$3^3 2 + 3^2 2^2 = 90$	$(3^3 2^2 + 3^2 2^3) P \left(\frac{N}{P}\right)^2$
total	108	$(180P + 27\ell) \left(\frac{N}{P}\right)^2$
MPI_ALL2ALLV ([9])	51	N

calls of the developed code. This observation is, in fact, deceptive, for the following reasons.

- The number of calls to MPI_GATHER/MPI_SCATTER can actually be reduced by a factor two if MPI_ALLGATHER is called in their stead, thus reducing the 108 to 63.
- The 18 calls to MPI_NEIGHBOR_ALLTOALLW can be contracted by a factor 3, thus reducing the number of calls to exactly 51.
- The call to MPI_ALLGATHER or MPI_GATHER/MPI_SCATTER is reportedly cheaper than a call to MPI_ALLTOALLV.
- the MPI_NEIGHBOR_ALLTOALLW is a *neighbor* collective, in that it involves only the 6 surrounding processes [65], and, as such, it is much cheaper than MPI_ALLTOALLV, which involves all processes in the communicator.¹²

Regarding the amount of data exchanged, in light of the fact that the ratio N/P can be held fixed to, say, the threshold τ , while the number of processes grows together with the mesh size, and that the number of layers ℓ is a small number only depending on the selected schemes, the total in Table 4.2 can be rewritten as

$$180\tau N + 27\ell\tau^2, \quad (4.1)$$

which is linear with N , even if considerably greater than.

¹²Even though one-dimensional sub-communicators can be used, thus reducing to P the number of processes participating to each call.

On the other hand, as long as the HPC technology advances, and bigger and bigger simulations are to be run, with the number of cells growing as N^3 , the number of maximum processes allowed by the present approach is asymptotically greater than that relative to a pencil-based decomposition. Indeed, for $\tau = 10$, the present solver allows $\frac{N^3}{1000}$ processors, which is more than N^2 (the processes allowed by the pencil based decomposition) for $N > 1000$.

The last, most important observation, is that Table 4.2 shows that the quantity of exchanged values, Eq. (4.1), varies with N only in consequence of the collectives `MPI_GATHER/MPI_SCATTER`, which are necessary to the SPIKE algorithm. If the truncated variant of the SPIKE algorithm, as described in Section 3.2, is implemented, then the number of exchanged values in step 2 and 4 at page 85 do not depend on P , and the total amount of exchanged data in Eq. (4.1) do not depend on N .

4.10 The Poisson equation for the pressure¹³

The Poisson equation stemming from the employed pressure-correction method, is discretized by means of the classic explicit second order finite differences (for the reasons explained in Section 6.1.1) and solved in the physical space, preferably through the so-called Scheduled Relaxation Jacobi method (SRJ) [66], which is select by the input line `ell_solver SRJ`. Such a method yields substantial reduction in the number of iterations needed for the iterative solution to converge, as compared to the classical Jacobi method (which is implemented in the code, and selectable through `ell_solver J`).

The way the divergence and gradient operators are built, is coherent with all other spatial operators throughout the code, therefore higher order formulæ can be used, in principle; on the other hand, the Laplace operator is derived as the matrix product between the \mathbf{B} matrices of the divergence and gradient operators, thus preventing the use of compact schemes (in that case, the matrices $\mathbf{A}^{-1}\mathbf{B}$ relative to the two operators should be explicitly computed; see Section 6.1.1 for details). The use of higher order explicit finite differences is possible, even though it is not tested, but the

¹³The Fortran module dealing with elliptic equations was not developed by the present author, who was involved only in so far as he defined the input/output variables: one rank-3 array in input containing the divergence, 1 rank-3 array in output containing the pressure.

specific schemes cannot be requested in the input file, since the identifying strings are hard-coded in the source files (as it is done for other operators, cf. Section [4.8.1](#)).

Chapter 5

Applications

In the following pages the CFD solver described in Chapter 4, which applies compact schemes in parallel through the SPIKE algorithm (Chapter 3) is used to solve the incompressible Navier-Stokes equations in various test-cases that are well-known in literature and studied worldwide in the fluid-dynamic community.

5.1 Set up

Continuity and momentum equations, in non-dimensional form, are the following

$$\begin{cases} \nabla \cdot \mathbf{V} = 0 & (5.1a) \\ \frac{\partial \mathbf{V}}{\partial t} + \mathbf{V} \cdot \nabla \mathbf{V} = -\nabla p + \frac{1}{\text{Re}} \nabla^2 \mathbf{V}. & (5.1b) \end{cases}$$

Eqs. (5.1a) and (5.1b) are solved with a standard pressure-correction scheme [67]: first, the current velocity \mathbf{V}^n is advanced without the contribution of the pressure term, so that a non-incompressible velocity field \mathbf{V}^* is obtained; then, the pressure is computed as the solution of the Poisson equation obtained by enforcing the incompressibility of $\mathbf{V}^{n+1} = \mathbf{V}^* - \nabla p$; finally, the ∇p is subtracted to \mathbf{V}^* to get \mathbf{V}^{n+1} .

5.2 Preliminary results

In this section the results are presented for a pair of two-dimensional flows, namely the Lid-Driven Cavity and Taylor-Green Vortex flows. For these two-dimensional tests, a simpler MATLAB script is used in place of the Fortran solver.¹ Originally, this code was a tool to the development of the SPIKE algorithm, as derived in Chapter 3, and in fact works as an emulator (actual decomposition, no actual parallelism); at the same time, it served as a tool for testing the compact schemes applied to the Navier-Stokes equations.

5.2.1 Two-dimensional lid-driven cavity flow

The laminar, incompressible flow in a square cavity, whose top wall moves tangentially with a uniform velocity, has served over and over again as a model problem for testing and evaluating numerical Navier-Stokes solvers [68]. Indeed, its importance comes from a number of interesting physical features peculiar to this test-case, which goes by the name of Lid-Driven Cavity flow. Among these, a primary vortex developing with increasing Reynolds number [69], an infinite sequence of viscous corner eddies at the two consecutive stagnation corners [70], as well as a particular singularity at the other two corners, where the moving wall meets with the two stationary side walls, [71–73].

The fluid-dynamic community clearly faced the two-dimensional variant of this test-case first, for which a wide *corpus* exists, today, that furnishes the basis for comparison; Ghia et al. [29] and Schreiber et al. [74] were among the first to publish benchmark data about it. The former is taken as reference.

The specific compact schemes for the spatial discretization are partly chosen consistently with Boersma [34] (choices from another reference where considered with caution, for being about compressible simulations [25]). Specifically, the stencils relative to each operator are reported in Figs. 5.1 to 5.6, where they are sketched close to the grid, to clarify how they are used. Concerning the coefficients, as explained in Section 2.2, they are relative to the maximum order of accuracy allowed on the stencil, unless otherwise specified by a number attached to the symbol.

¹As explained in Section 4.5, the program cannot run with less than two processes per direction; in turn, a true two-dimensional run is not possible, since it would require the use of one single cell along the spanwise direction.

Table 5.1: Values of the stream function on the streamlines depicted in Figs. 5.7 to 5.9. Extracted from [29, Tab. III, p. 403].

Contour letter	Value of ψ		Contour #	Value of ψ
a	-1	$\times 10^{-10}$	0	1×10^{-8}
b	-1	$\times 10^{-7}$	1	1×10^{-7}
c	-1	$\times 10^{-5}$	2	1×10^{-6}
d	-1	$\times 10^{-4}$	3	1×10^{-5}
e	-1	$\times 10^{-2}$	4	5×10^{-5}
f	-3	$\times 10^{-2}$	5	1×10^{-4}
g	-5	$\times 10^{-2}$	6	2.5×10^{-4}
h	-7	$\times 10^{-2}$	7	5×10^{-4}
i	-9	$\times 10^{-2}$	8	1×10^{-3}
j	-1	$\times 10^{-1}$	9	1.5×10^{-3}
k	-1.1	$\times 10^{-1}$	10	3×10^{-3}
l	-1.15	$\times 10^{-1}$		
m	-1.175	$\times 10^{-1}$		

The simulation has been driven forward to the steady state by means of the explicit Euler method, for the two Reynolds numbers $Re = 1000$ and $Re = 3200$, on a grid of 128×128 cells, according to [29]; for the former Reynolds number the grid 64×64 was used as well. The results are presented in terms of streamlines and velocity profiles.

In Figs. 5.7 to 5.9, the contour plots of the stream function, as obtained with the MATLAB code, are plot on the left, as compared to those from the reference [29, pg. 400-402], on the right, which were obtained with a multigrid method. The curves are plot at the same levels used in the reference (see Table 5.1 copied from [29, Tab. III, p. 403]). The typical separations and secondary vortices at the bottom corners of the cavity (for both $Re = 1000$ and $Re = 3200$) as well as at the top left (for $Re = 3200$ only) can be seen. The streamlines show an excellent agreement, especially considering the extension of secondary vortices in both x - and y -direction, with the cited benchmark, and other established results [20, 75–77], thereby confirming that the present method yields quantitatively accurate solutions.

The vertical component of velocity on the horizontal center-line, and the horizontal component on the vertical center-line are plotted in

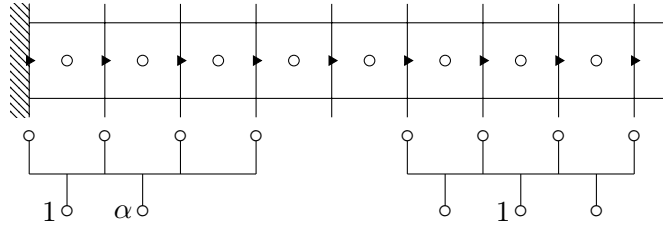


Figure 5.1: Stencils of the central and boundary schemes used in the interpolation of u^2 from u -nodes to p -nodes. The latter is used with $\alpha = 0$.

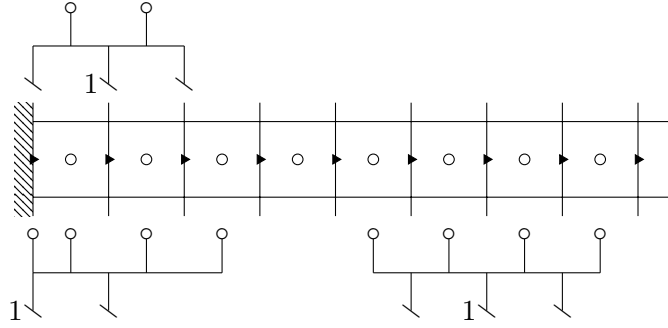


Figure 5.2: Stencils of the central, near-boundary and boundary schemes used in the differentiation of u^2 from p -nodes back to u -nodes.

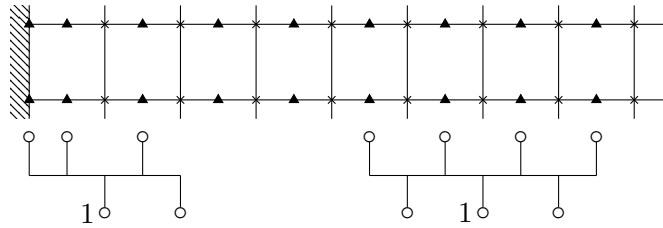


Figure 5.3: Stencils of the central and boundary schemes used in the interpolation of v from v -nodes to corners.

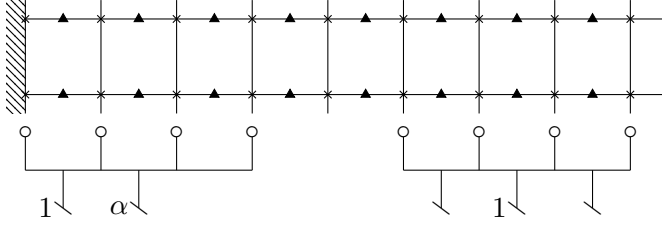


Figure 5.4: Stencils of the central and boundary schemes used in the x -differentiation of $\bar{u}\bar{v}$ from corners back to v -nodes. $\alpha = 0$ is chosen for this case.

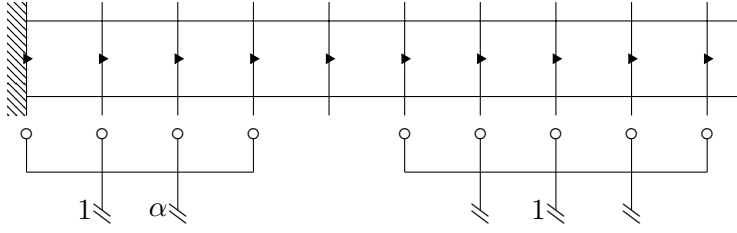


Figure 5.5: Stencil of central and boundary schemes used to perform the second x -derivative of u . Note that the first node is *not staggered*, as opposed to Fig. 5.6. $\alpha = 1$ is set.

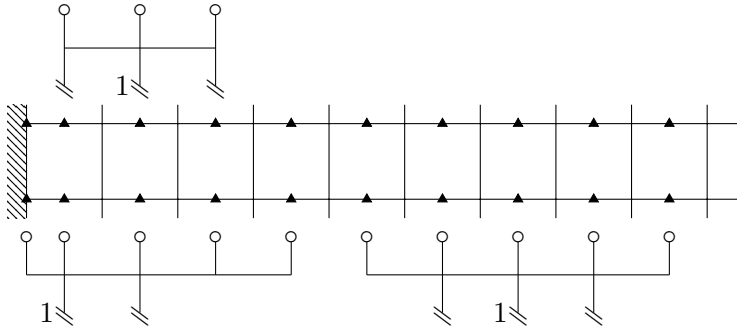


Figure 5.6: Stencil of central, near-boundary and boundary schemes used to perform the second x -derivative of v . Note that the first node is *staggered*, as opposed to Fig. 5.5.

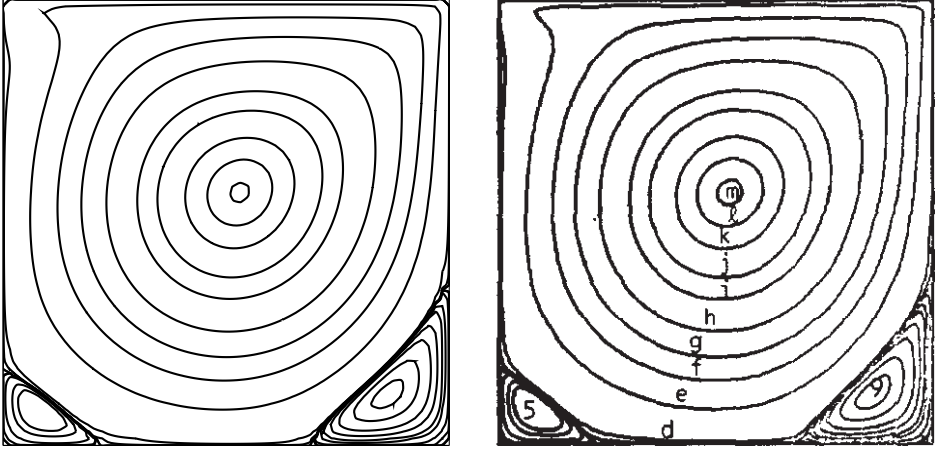


Figure 5.7: Contour plot of the stream function, at $Re = 1000$, for the values reported in Table 5.1. *Left:* Computed on the uniform grid 64×64 . *Right:* [29, p. 400] on a 129×129 uniform grid.

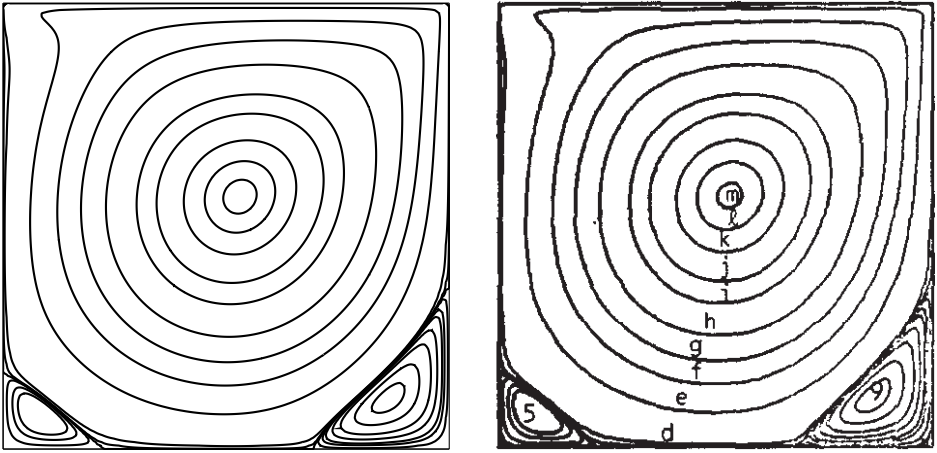


Figure 5.8: Contour plot of the stream function, at $Re = 1000$, for the values reported in Table 5.1. *Left:* Computed on the uniform grid 128×128 . *Right:* same as in Fig. 5.7.

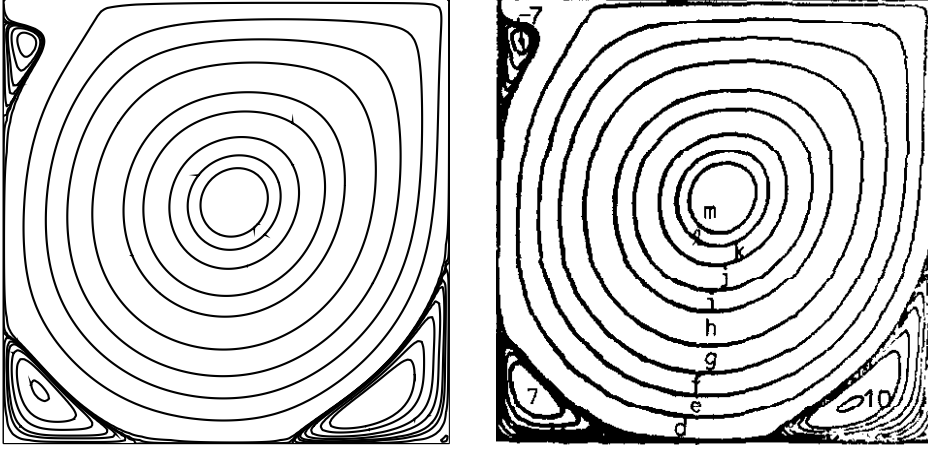


Figure 5.9: Contour plot of the stream function, at $Re = 3200$, for the values reported in Table 5.1. *Left:* Computed on the uniform grid 128×128 . *Right:* [29, p. 401] on a 129×129 uniform grid.

Figs. 5.10a and 5.10b for $Re = 1000$, and in Fig. 5.10c for $Re = 3200$, as compared to the values collected in [29, Tab. I-II, p. 398-399]. Note that velocity profiles obtained on the grid 128×128 match very well with these referenced values, which were obtained on a grid 256×256 . This behavior is in common with the method tested in [31].

The near-linearity of the velocity profiles in the central core of the cavity is indicative of the uniform vorticity region that develops here for high Reynolds numbers [69]. In the near-wall regions, on the contrary, these velocity profile should join those of the walls themselves, thus exhibiting high gradients in developing the boundary layers. In particular, the boundary layer on the top is clearly thinner than that on the bottom wall: $\delta_t \approx 0.2$ and $\delta_b \approx 0.4$ for $Re = 1000$. This is due to the difference in the local Reynolds number (based on the local velocity), that is higher for the top wall, which entrains the fluid with its velocity $U_{lid} = 1$, whereas it is lower on the bottom, which brakes the velocity to zero. Clearly, all boundary layers get thinner as the Reynolds number increases (cf. the case $Re = 3200$).

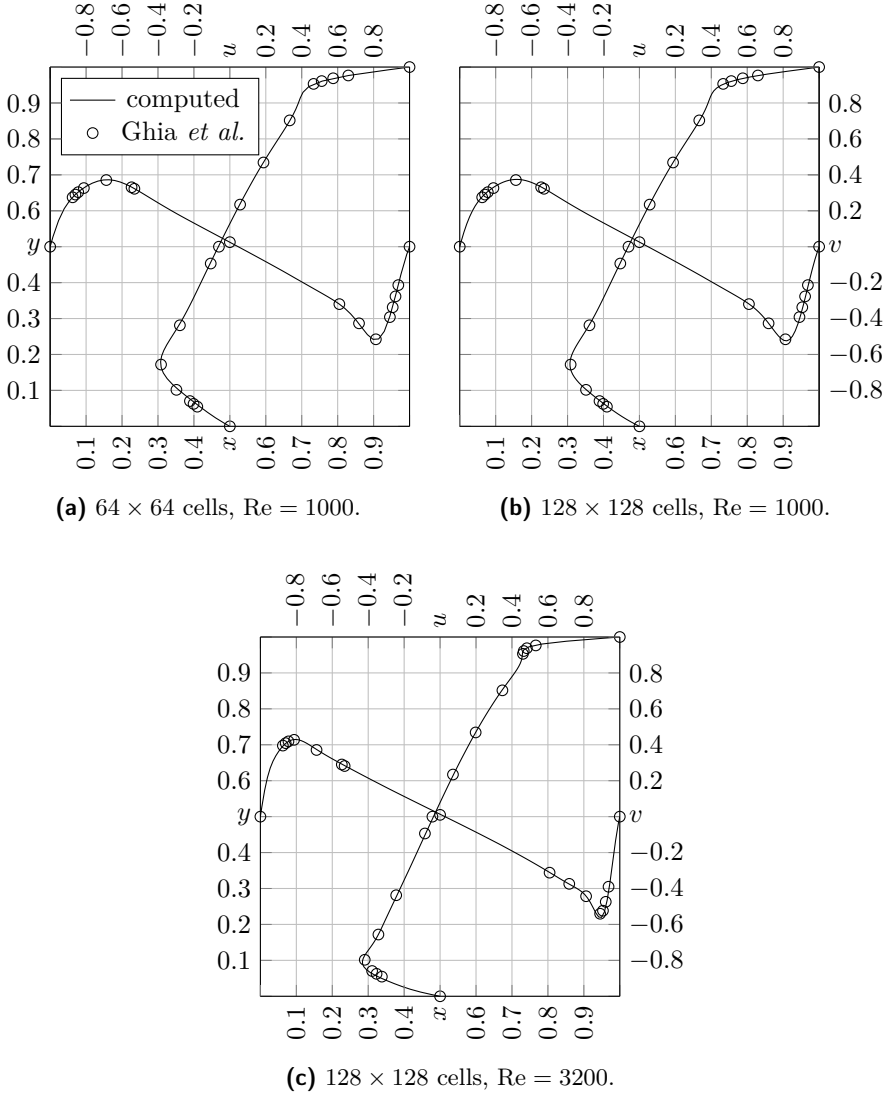


Figure 5.10: Profiles $u(1/2, y, 1/2)$ and $v(x, 1/2, 1/2)$ for different combinations of Reynolds number and spacing, as compared to the reference results [29, Tab. I-II, pp. 398-399], obtained on a 129×129 uniform grid.

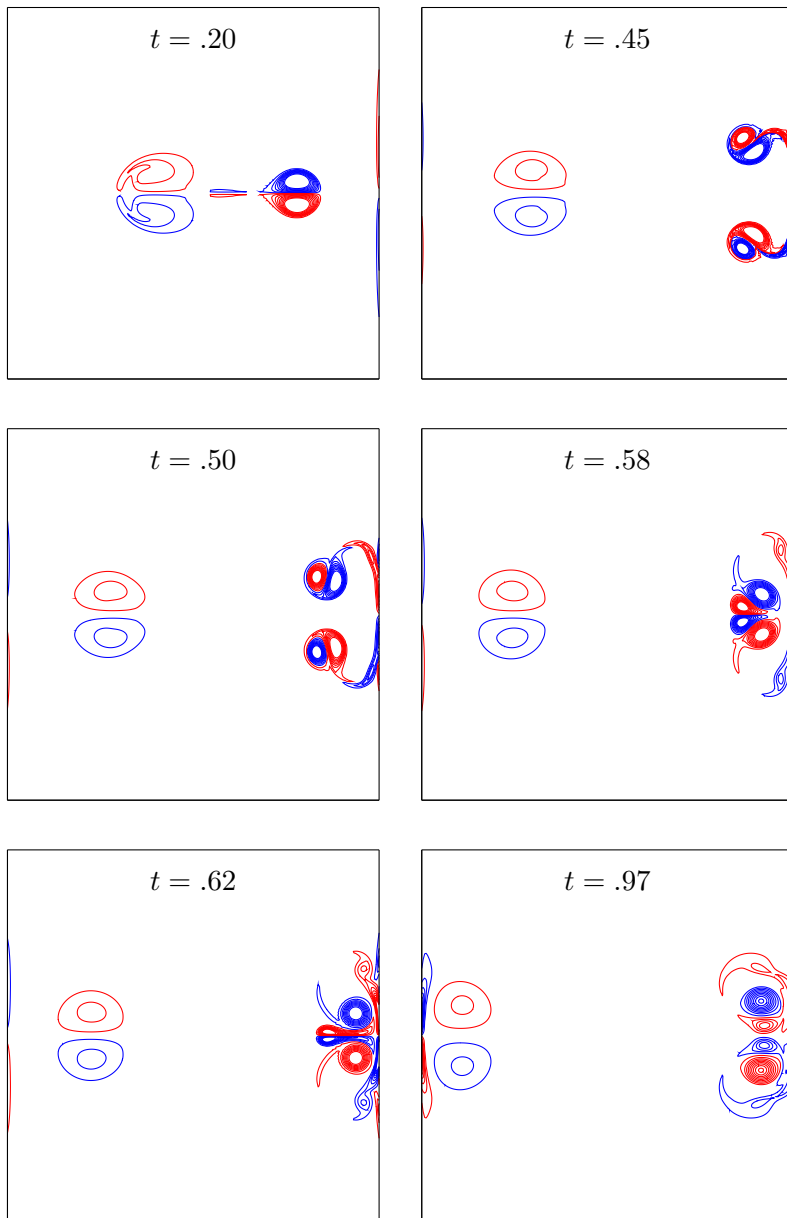


Figure 5.11: Dipole-wall interaction: vorticity contours at selected time instances for $\text{Re} = 2500$.

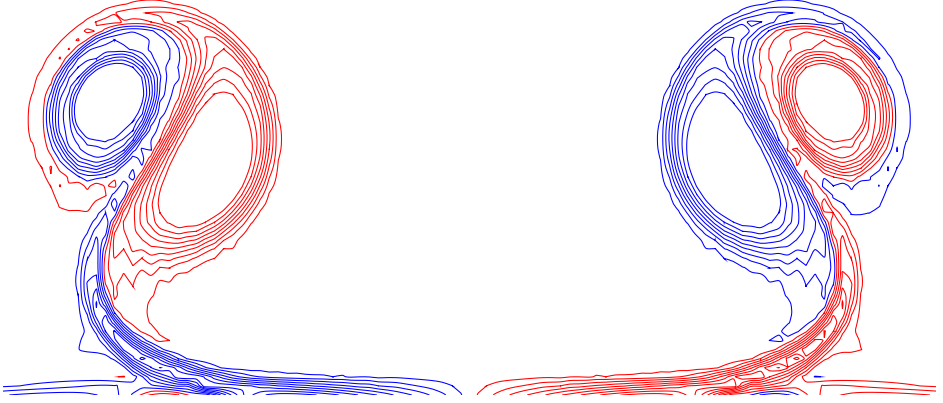


Figure 5.12: Detailed zoom of the collision zone at $t = 0.45$. The occurrence of grid-to-grid oscillations is coherent with the observation made in Section 5.4 with reference to Fig. 5.18.

5.2.2 Dipole-wall interaction

The simulation in this section are performed with the same MATLAB script used in Section 5.2 to emulate the Fortran code in two dimensions.

Despite the dipole-wall collision could seem a simple academic test-case at first sight, it is, on the contrary, of relevant importance in DNS, for the reasons explained in the following. Direct numerical simulations of turbulence in bounded domains have recently elucidated the importance of the role of no-slip boundaries in general, and vortex-wall interactions in particular. Indeed, the boundaries act as a source of relatively small-scale vortices. In an attempt to quantify the amount of vorticity produced near the no-slip walls Clercx and Heijst [78] proposed to set up a relatively simple numerical experiment: create a self-propelling dipole which travels throughout a square container with no-slip walls, eventually hitting a boundary, and analyze the vorticity production in the boundary layers. After the formation of the boundary layers and the subsequent detachment, a complicated sequence of vortex-wall interactions is observed to take place. Such events have been reported already one decade ago [79].

The simulations starts from the initial conditions imposed as a dipole made up of a pair of two vortices,

$$\omega_{1,2}(x, y, 0) = \pm\omega_e \left[1 - \left(\frac{r_{1,2}}{r_0} \right)^2 \right] e^{-\left(\frac{r_{1,2}}{r_0} \right)^2},$$

where $r_1 = \sqrt{(x - x_1)^2 + (y - y_1)^2}$ and $r_2 = \sqrt{(x - x_2)^2 + (y - y_2)^2}$ are the distances from the centers of the vortices located at $(x_1, y_1) = (0, +0.1)$ and $(x_2, y_2) = (0, -0.1)$, the vortex radius r_0 is assumed equal to 0.1, and the maximum vorticity ω_e equal to 300. These initial condition is imposed in terms of velocity component, that is

$$\begin{aligned} u(x, y, 0) &= -\frac{1}{2}\omega_e(y - y_1)e^{-\left(\frac{r_1}{r_0}\right)^2} + \frac{1}{2}\omega_e(y - y_2)e^{-\left(\frac{r_2}{r_0}\right)^2} \\ v(x, y, 0) &= +\frac{1}{2}\omega_e(x - x_1)e^{-\left(\frac{r_1}{r_0}\right)^2} - \frac{1}{2}\omega_e(x - x_2)e^{-\left(\frac{r_2}{r_0}\right)^2}. \end{aligned}$$

The simulation has been conducted to a final time $t = 1.5$ in a square domain discretized by a 256×256 grid. The Reynolds number, $\text{Re} = 2500$, is based on the maximum velocity in the initial condition, $U_{\max} \approx 10$, and on distance D between the centers of the two monopoles at time $t = 0$ ($D = y_1 - y_2 = 0.2$). A sample evolution of the vorticity field is presented in Fig. 5.11 showing the instantaneous solutions at selected time moments. Fig. 5.12 depicts a detailed zoom of the collision zone at $t = 0.45$.



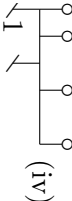
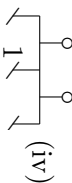
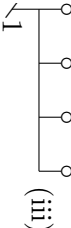
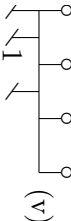
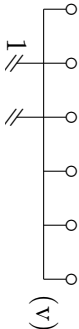
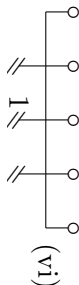
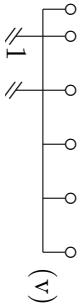
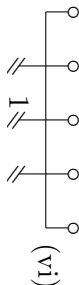
5.3 Three-dimensional cavity at $\text{Re} = 1000$ and $\text{Re} = 3200$

The results shown in the present section and in Section 5.4 are carried out by means of the Fortran code described in Chapter 4. All the used schemes are collected in Table 5.2 according to the rules defined in Section 2.2, and the order of accuracy is indicated with a roman number beside each sketch. (Clearly, for periodic case, e.g., the Taylor-Green Vortex test-case of Section 5.4, the “central scheme” column only is of interest.)

The well-defined structures characterizing its flow, combined with the very simple geometry, make the investigation of a three-dimensional lid-driven cavity flow a useful benchmark for numerical schemes. The geometric simplicity allows to evaluate the efficiency of the numerical scheme as it is, without the introduction of complex transformations which could affect the overall accuracy.

Despite the three-dimensional cavity flow calculations were first carried out in the late seventies [80, 81], only one decade later the results of comprehensive studies on the matter were collected by Deville et al. [82], with the target of describing the formation and evolution of the three-dimensional structures characterizing the flow. However, the results

Table 5.2: Left-boundary and central schemes used for the lid-driven cavity test-case (right schemes are symmetric to left schemes); the central schemes are used for the Taylor-Green Vortex test-case as well. The order of accuracy is the roman number in parenthesis beside the sketch. Note: the outermost interpolation sketch merely indicates an assignation equation for the variable at the boundary; the near-boundary interpolation scheme from centers to faces is the symmetric 4th order scheme with the leftmost value considered known. The schemes for the first derivative consider the value on the boundary unknown, since it is needed for the computation of the convective term.

ord. of deriv.	type	outermost to innermost non-central schemes		central scheme	
0	c2f				
	f2c				
1	c2f		(iv)		(iv)
	f2c		(iii)		(v)
	c2c		(v)		(vi)
	f2f		(v)		(vi)

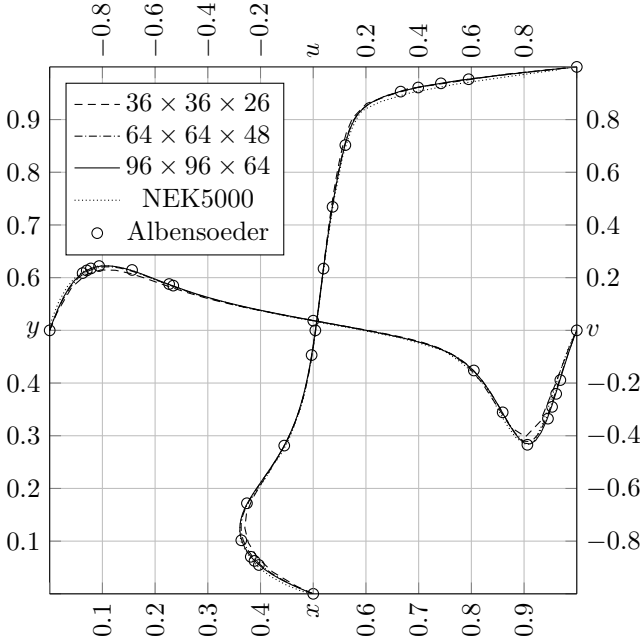


Figure 5.13: Profiles $u(1/2, y, 1/2)$ and $v(x, 1/2, 1/2)$ for $Re = 1000$ obtained on three level of refinement, as compared to reference results [68], as well as those obtained with NEK5000.

for $Re > 1000$ (e.g., $Re = 3200$) were inconclusive, due to unacceptable method- and mesh-dependencies. The first correct three-dimensional linear stability analysis for the two-dimensional cavity flow (with periodic boundary conditions along the spanwise direction), as well as an accurate study of the full three-dimensional variant, were carried out by Albensoeder and Kuhlmann [68, 83], barely a decade ago.

From a physical point of view, this flow exhibits a primary stationary vortex (common to the 2D cavity), which is disturbed by the appearance of secondary three-dimensional structures, whose presence forbids a two dimensional flow; such vortices become unsteady at moderately high Reynolds number, and are known as Taylor-Görtler-like (TGL) vortices.

A first set of simulations is performed at $Re = 1000$ for three levels of refinement of the grid, with a unitary CFL number, whereas one simulation is performed at $Re = 3600$ on the single meshgrid $80 \times 80 \times 80$. In both cases, 6th order compact schemes are used in the center, whereas lower

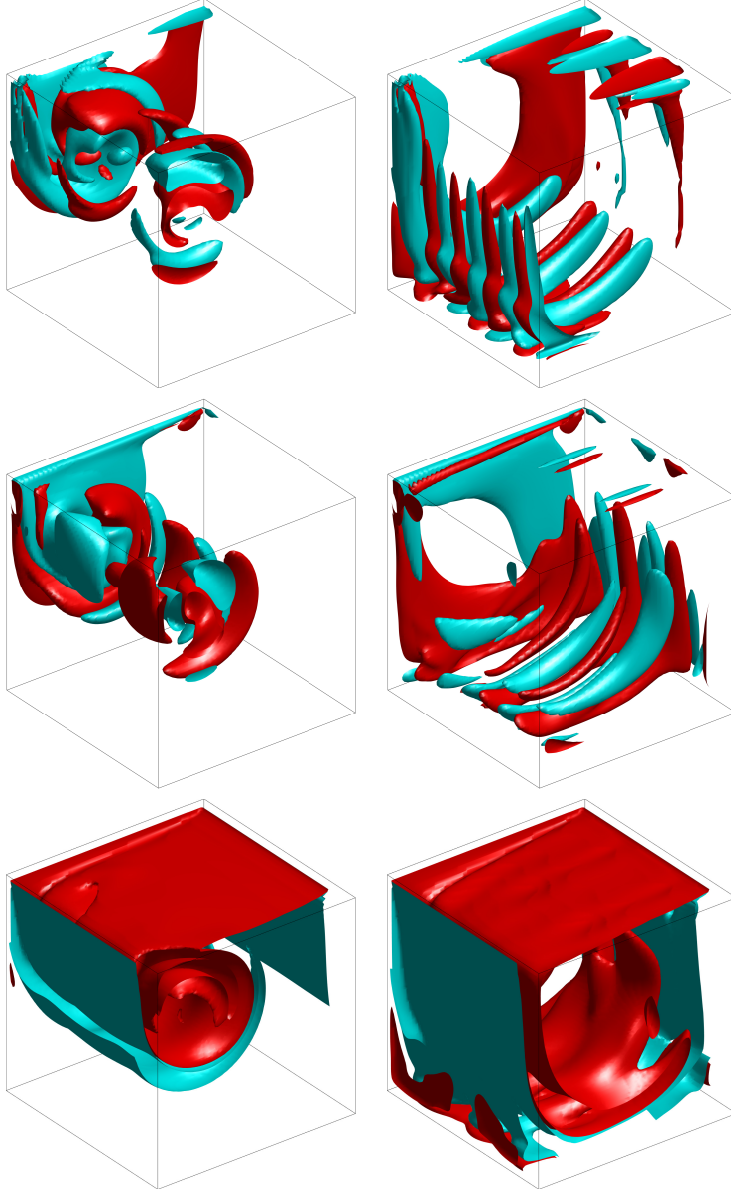


Figure 5.14: Isosurfaces of vorticity components a little time after start-up (left), and after the onset of TGL vortical structures (right); from top to bottom: $\Omega_x = \pm 1$, $\Omega_y = \pm 1$, $\Omega_z = \pm 1$. $Re = 3200$. (NOTE: A few slices of data in the front-right face are removed to allow a clear view of the inner structures.)

order formulæ are used at the boundary, as well as close to it.

In Fig. 5.13 the velocity profiles for $\text{Re} = 1000$ are compared to the reference [68], in which the simulation was carried out by means of a Chebyshev-collocation method applied on the grid $96 \times 96 \times 64$; moreover, Fig. 5.13 includes profiles computed in NEK5000 on a grid of $12 \times 12 \times 8$ spectral elements, each of which is $8 \times 8 \times 8$ cells, thus yielding the same total number of cells relative to the finest used resolution, $96 \times 96 \times 64$. Despite all methods lead to the same, correct result in the inner part of the cavity, they differ close to the boundaries, where the present method's agreement with the reference [68] is superior to NEK5000's, even on the medium grid, $64 \times 64 \times 48$.

The result for $\text{Re} = 3200$ is shown in Fig. 5.14 in terms of iso-surfaces, for values ± 1 , of the vorticity components Ω_x , Ω_y , and Ω_z . The main and secondary vortices, which are common to the two-dimensional cavity, are apparent from Ω_z , shortly after the simulation has started; after the initial start-up phase, the TGL vortical structures are clearly visible from both Ω_x and Ω_y 's iso-surfaces.

5.4 Taylor-Green Vortex at $\text{Re} = 1600$

A classic test-case of relevant importance is the viscous Taylor-Green Vortex flow (TGV), which is defined in both 2D and 3D (within domains $[2\pi \times 2\pi]$ and $[2\pi \times 2\pi \times 2\pi]$ respectively). The initial condition in 2D is

$$\begin{aligned} u_0(x, y) &= -\sin(kx) \cos(ky) \\ v_0(x, y) &= +\cos(kx) \sin(ky), \end{aligned}$$

and the analytical solution is the exponential decay of this initial condition, through the function $e^{-\frac{2k^2}{\text{Re}}t}$.²

A similar solution, through the coefficient $e^{-\frac{3k^2}{\text{Re}}t}$, exist for the three-dimensional case, which has the following initial condition [84] (often used

²The index k is such that $2k$ is the number of vortices per side; the same k is chosen for both x - and y -side, for simplicity.

with $\theta = 0$ [85]),

$$\begin{aligned} u_0(x, y, z) &= \frac{2}{\sqrt{3}} \sin\left(\theta + \frac{2\pi}{3}\right) \sin(kx) \cos(ky) \cos(kz) \\ v_0(x, y, z) &= \frac{2}{\sqrt{3}} \sin\left(\theta - \frac{2\pi}{3}\right) \cos(kx) \sin(ky) \cos(kz) \\ w_0(x, y, z) &= \frac{2}{\sqrt{3}} \sin(\theta) \cos(kx) \cos(ky) \sin(kz) \end{aligned}$$

but is never observed, since the flow unconditionally undergoes the transition to non-isotropic turbulence, which evolves towards decaying, homogeneous, isotropic turbulence.

The importance of the (three-dimensional) TGV flow lies exactly in the time evolution of the initially organized structures. Indeed, the transition to increasingly smaller, but still organized, scale vortices, is followed by vortex-stretching mechanisms [84], which lead to the formation of fully developed, disorganized, decaying worm-shaped vortices, which characterize the developed turbulence. Afterward, the motion is damped down by viscosity, whose action is stronger on these newly formed smaller scales, until the flow is finally steady. Based on direct numerical simulations conducted in the incompressible regime, the existence of a fairly consistent dissipation peak at the non-dimensional time $t = 9$ is reported for $Re = 800, 1600, 3000$ and 5000 [84, 86].

The conventional wisdom is that numerical diffusion affecting the initial convection stage is undesirable and should be avoided; indeed, the kinetic energy can be damped only by the viscous effect of the resolved (as well as modelled, in LES) scales, and it should otherwise be conserved [38]. According to this consensus, the time-evolution of integral quantities has been widely used as reference to assess the unwanted numerical dissipation effects.

This test-case has thus served to prove the conservation properties of the developed CFD solver, as compared to the reference [87]. At the same time, it is used to assess the scaling of the program in its current state (see Table 5.3 and Figs. 5.21 and 5.22). Concurrently, it also served to verify that the parallelization has no impact on the results, nor has the restarting procedure (compare matching plots of Figs. 5.16 and 5.17).³

³Differences consequent to a different choice of the number of processes can occur in the case that the pressure-related convergence condition is set `.TRUE.` by the first process hitting the threshold (which is not the case, at the moment). Differences between

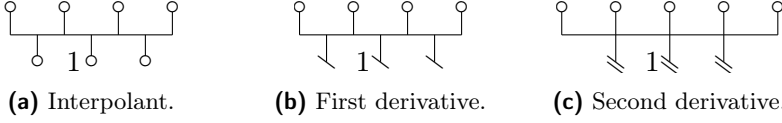


Figure 5.15: Sketches relative to the 6th order compact schemes discretizing the operators used for the TGV simulations.

Due to the periodic nature of this test-case, only one symmetric scheme is required for each interpolation/differentiation operator. The spatial derivatives of order 0, 1, and 2, are discretized with the 6th order compact operators whose stencils are sketched in Fig. 5.15, according to the rules listed in Section 2.2. The classic 4th order Runge-Kutta method (RK4) is used for the time integration.

Figs. 5.16 and 5.17 show the dissipation time history for several mesh sizes and process-grid sizes, as compared to the reference solution obtained with pseudo-spectral methods on a $512 \times 512 \times 512$ mesh [87]. At the resolution of $256 \times 256 \times 256$, which means one eighth the number of points used in the reference solution, the curve obtained with the present CFD solver stays stuck to the reference curve up to $t = 10$ and beyond. The solver still behaves well with roughly 5% as many points as the reference grid ($192 \times 192 \times 192$), on which the dissipation is accurate up to $t = 8$. This is an indication that the present DNS solver can be promisingly upgraded by including an LES module.

In Fig. 5.18 a contour plot compares the module of vorticity to that presented in the same work [87]: the overall pattern is well captured by the code.

Finally the iso-surfaces of the vorticity vector's z -component relative to two values are shown in Figs. 5.19 and 5.20 at different times.

5.4.1 Scalability assessment

Being $T(n, p)$ the wall clock time relative to the solution of the problem of *total* size n , as obtained with p processes, the speed-up is defined as the ratio between serial and parallel wall clock times,

$$S(n, p) = \frac{T(n, 1)}{T(n, p)}, \quad (5.2)$$

restarted and non-restarted runs occur as a consequence of the first guess for the pressure field. In both cases the differences are close to the tolerance on the Poisson equation.

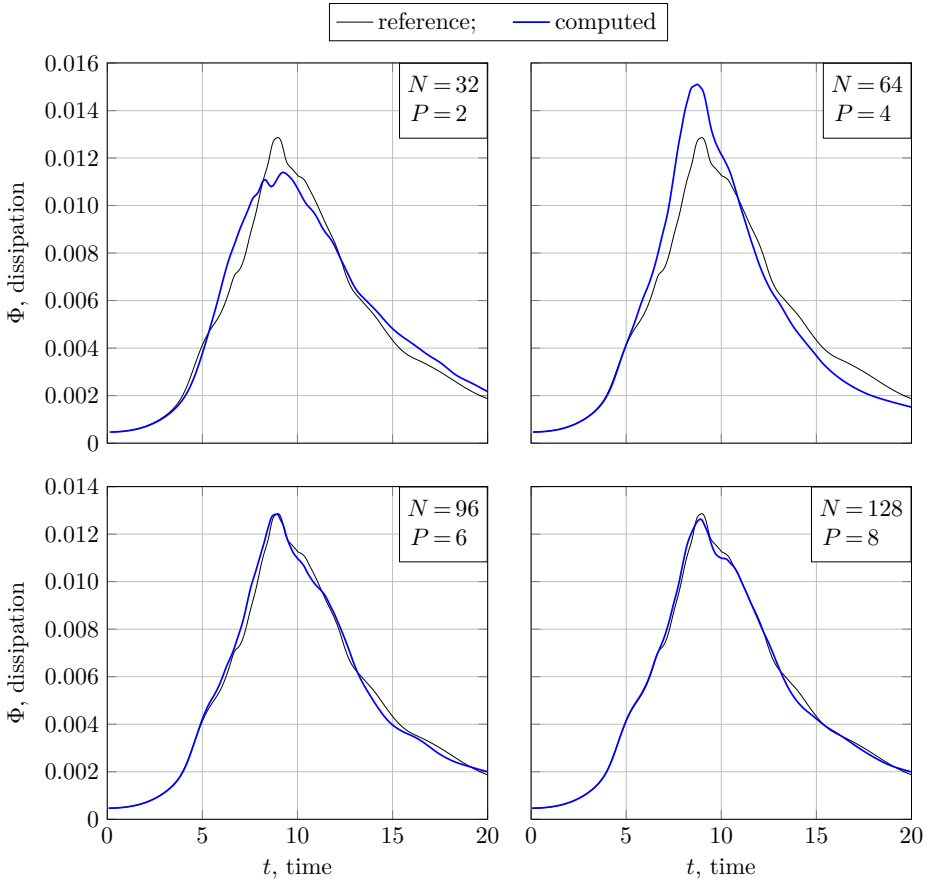


Figure 5.16: Dissipation time evolutions for different meshes, as compared to the spectral reference solution, obtained on a $512 \times 512 \times 512$. The solutions are computed on an MPI cartesian grid of $P \times P \times P$ processes, so that $N/P = 16$, where $N \times N \times N$ is the computational grid, specified in the plots.

whose ideal value is exactly p , in which case it is referred to as *linear speed-up*, rarely achieved in real-world applications. A far more meaningful quantity, widely used as index of parallel performance, is the parallel efficiency, or “per process” speed-up [88], defined as

$$E(n, p) = \frac{S(n, p)}{p} = \frac{T(n, 1)}{p \times T(n, p)}, \quad (5.3)$$

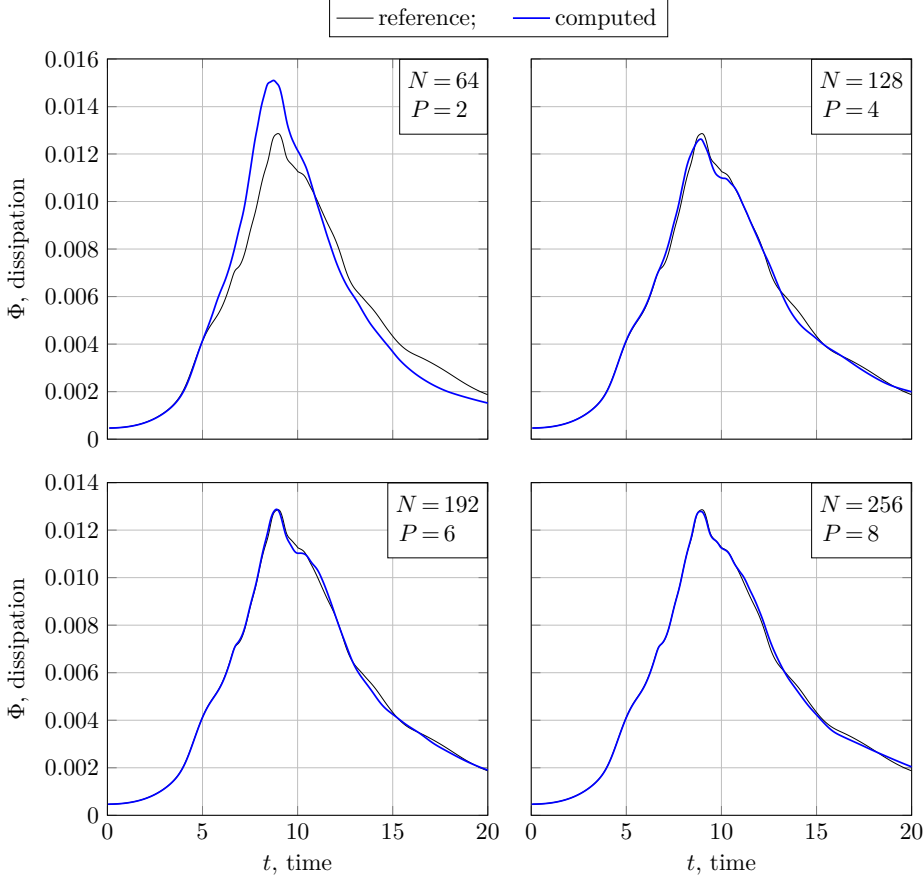


Figure 5.17: Dissipation time evolutions for different meshes, as compared to the spectral reference solution, obtained on a $512 \times 512 \times 512$. The solutions are computed on an MPI Cartesian grid of $P \times P \times P$ processes, so that $N/P = 32$, where $N \times N \times N$ is the computational grid, specified in the plots.

whose ideal value is 1, and corresponds to the linear speed-up.⁴ In this case, where the scalability is assessed by incrementing p while keeping n fixed, we speak of *strong* scalability.

When the total problem size is increased at the same rate as p , so that the computational load per process $n_p = n/p$ is kept constant, *weak*

⁴Given its definition, $E(n, p)$ can also be regarded as the serial-to-parallel ratio between the *cumulative* wall clock times.

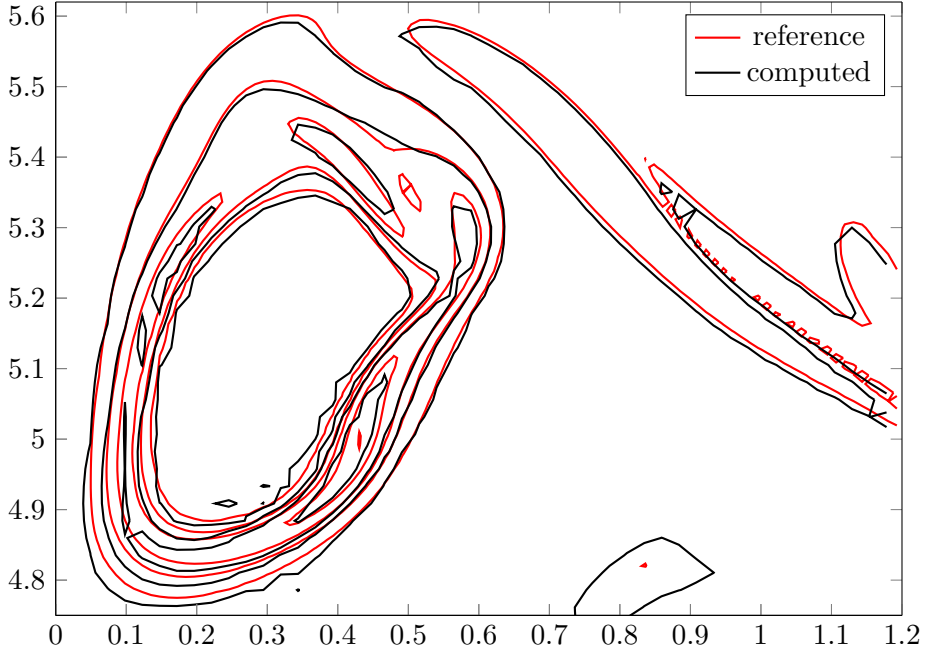


Figure 5.18: Vorticity contours in the $x = 0$ plane of the Taylor-Green vortex at $\text{Re} = 1600$ obtained with the present code on a $256 \times 256 \times 256$ grid (black), as compared to the reference [87]. The contours are plotted for several values of $|\Omega| = |\nabla \times \mathbf{V}|$, namely 1, 5, 10, 20 and 30.

scalability is measured. In this case, the speed-up can be written as

$$S(n_p \times p, p) = \frac{T(n_p, 1)}{T(n_p \times p, p)}, \quad (5.4)$$

and its ideal value is 1, a value fairly easier to approach than for strong scalability (hence the adjectives *strong* and *weak*).

It is crucial to note that, with reference to the three-dimensional domain, $n = N^3$ and $p = P^3$ in Eqs. (5.2) to (5.4), being N and P the number of *total* points per direction and the number of processes per direction (the same along the three directions, for simplicity).

With reference to the present case, two remarks are of importance.

- The runs were aimed at assessing the *weak* scalability for three distinct values of the load per process; the timings of few of these are also used to give a rough estimate of the *strong* scalability.

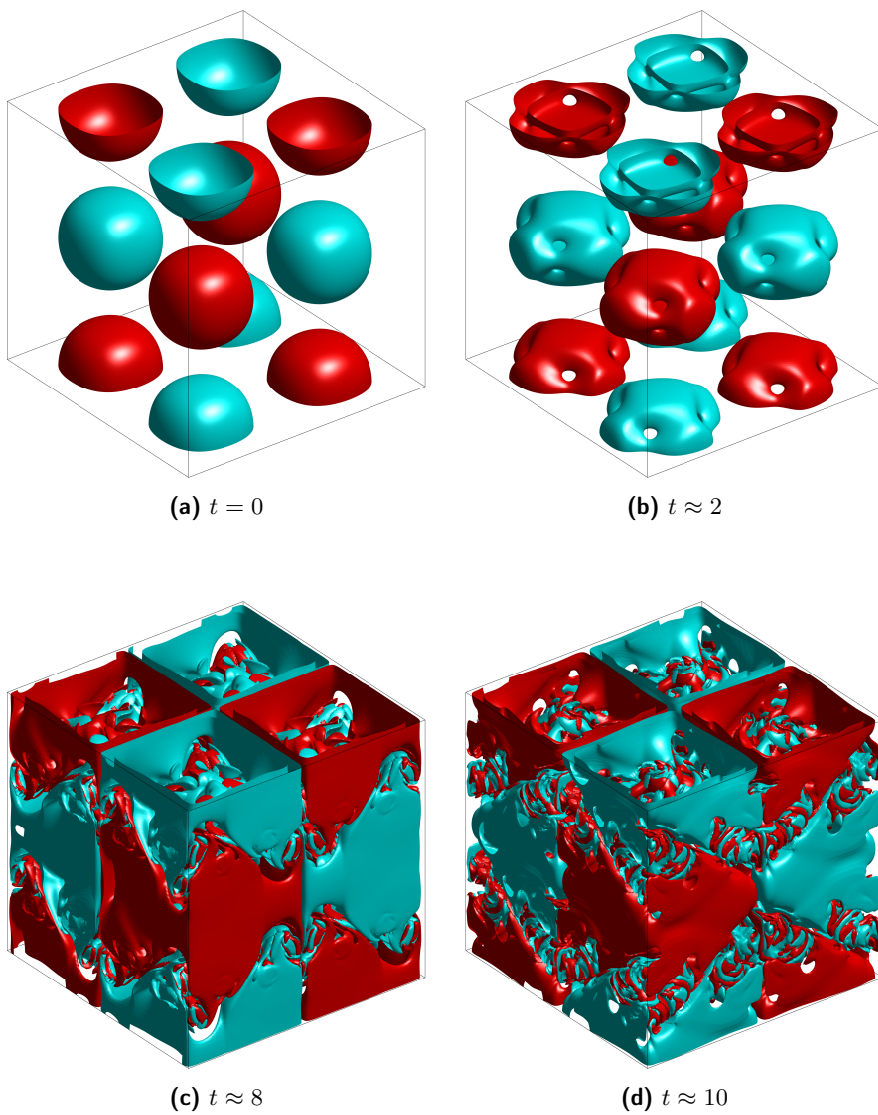


Figure 5.19: Iso-surfaces of vorticity for the Taylor-Green vortex test-case relative to $\Omega_z = \pm 1$ at various times. Grid $256 \times 256 \times 256$.

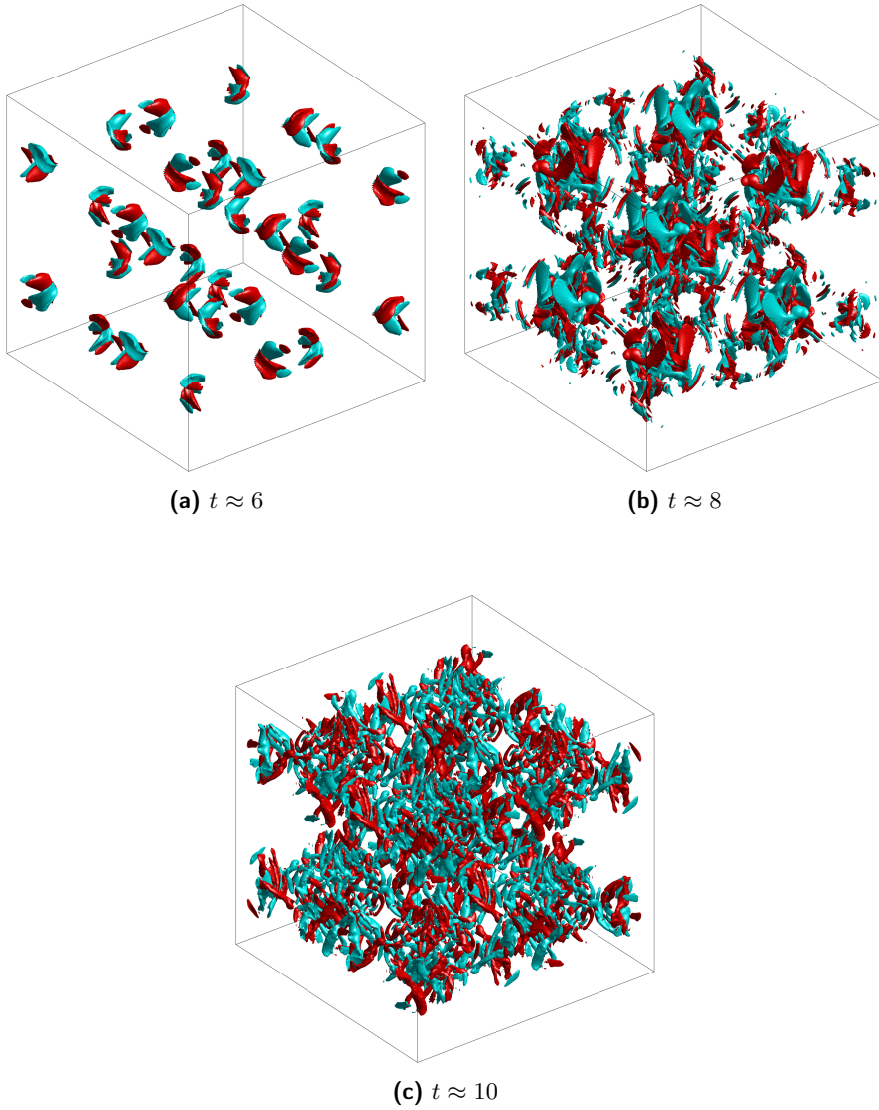


Figure 5.20: Iso-surfaces of vorticity for the Taylor-Green vortex test-case relative to $\Omega_z = \pm 7$ at various times. Grid $256 \times 256 \times 256$.

- At the time of writing, the code cannot run with less than 2 processes per direction, thus precluding a serial run. The assumption is made that the run with 2^3 processes ($2 \times 2 \times 2$ grid) has unitary speed-up when the *weak* scaling is concerned, whereas the configurations with 2^3 , 3^3 , and 4^3 have unitary efficiency when the *strong* scaling is to be assessed on total grids of 128^3 , 192^3 , and 256^3 cells.
- Since the same constant CFL number is used in all simulations ($C = a \frac{\Delta t}{\Delta x} = \frac{1}{2}$), the weak scaling implies a decrease in Δt , as a consequence of the increase in the *one-dimensional* total problem size N ($\Delta t \propto \Delta x = \frac{L}{N}$, being L the domain length), and an increase in wall clock time needed to reach the same, fixed final time of the simulated phenomenon.

Concerning the latter observation, it is clear that the wall clock time grows with N , *regardless* of the parallel framework, and therefore the corresponding expected increase should be ruled out. In other words, in order to assess the speed-up, a normalized wall clock time must be defined, and used in place of the actual one. To do so, it is fruitful to relate P and Δt , by manipulating the mesh spacing Δx and the CFL number C , thus obtaining

$$P\Delta t = \frac{LC}{aQ} = \text{constant},$$

from which it is clear that T can be normalized either by multiplying it by Δt or dividing it by P , and the latter choice is made here,

$$\tilde{T}(N^3, P^3) = \frac{T(N^3, P^3)}{P} \quad (5.5)$$

(Clearly $\tilde{T} = T$ for $P = 1$.) The definition of speed-up in Eq. (5.4) is thus adapted to the present needs as follows,

$$\tilde{S}(N^3, P^3) = \frac{\tilde{T}(Q^3, 1)}{\tilde{T}(N^3, P^3)} = \frac{P \times T(Q^3, 1)}{T(N^3, P^3)}. \quad (5.6)$$

The wall clock times relative to three values of load per process, 16, 32 and 64, are collected in Table 5.3, together with the speed-up, in bold; the unitary speed-up for $P = 2$ is relative to the assumed perfect scaling with respect to the serial case (which cannot be run, see Section 6.3.1); these value of the speed-up are plotted against P^3 in Fig. 5.21.

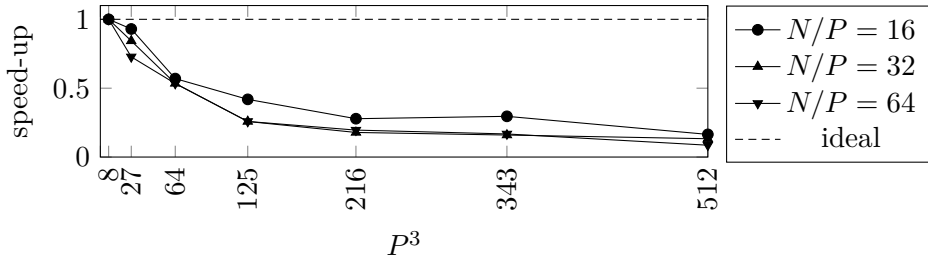


Figure 5.21: Speed-up relative to the weak-scaling, as defined in Eq. (5.4) with the wall normalized clock time of Eq. (5.5), for three values of load per process.

Table 5.3: Weak scaling, for three values of $(\frac{N}{P})^3$. Data from color-coded rows are used to obtain correspondingly colored curves in Fig. 5.22.

$\frac{N}{P}$	N	P	P^3	$T(N^3, P^3)$ (min)	$\tilde{T}(N^3, P^3)$	Speed-up
16	32	2	8	8.14	4.07	1
	48	3	27	13.1	4.38	0.930
	64	4	64	28.6	7.14	0.570
	80	5	125	48.6	9.71	0.419
	96	6	216	87.7	14.6	0.278
	112	7	343	96.4	13.8	0.296
	128	8	512	198	24.7	0.165
32	64	2	8	111	55.6	1
	96	3	27	198	65.9	0.843
	128	4	64	416	104	0.535
	160	5	125	1080	215	0.258
	192	6	216	1870	311	0.179
	224	7	343	2440	348	0.160
	256	8	512	3340	418	0.133
64	128	2	8	2250	1120	1
	192	3	27	4640	1550	0.727
	256	4	64	8430	2110	0.534
	320	5	125	21 800	4350	0.258
	384	6	216	34 500	5750	0.196
	448	7	343	47 000	6710	0.168
	512	8	512	105 000	13 100	0.0860

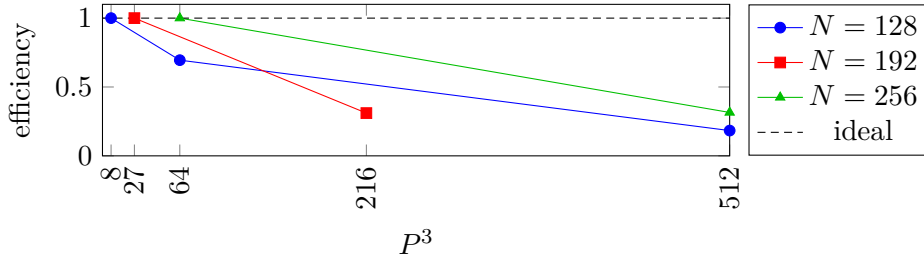


Figure 5.22: Efficiency relative to the strong-scaling, as defined in Eq. (5.3) with the normalized wall clock time of Eq. (5.5), for three total domain sizes.

Seven rows in Table 5.3 are highlight with three colors; the rows of common color are relative to a fixed total domain size and, as such, can be used to give a rough estimate of the strong scaling; the parallel efficiency is indeed plotted in Fig. 5.22, using the same color-coding of Table 5.3.

Chapter 6

Conclusions and future works

In this chapter, possible opportunities to improve the existing code, fix problematic shortcomings, and, above all, get a faster code, are briefly discussed, together with the verified or supposed causes of deficiencies.

6.1 CFD-related

6.1.1 The Poisson equation for the pressure

The solution of the Poisson equation is notoriously the bottleneck of the CFD solvers (and as such, it has driven the research towards new approaches [9, 34, 42, 89–91]), and the present code is no exception.

Considering the difficulties inherent in the development of the CFD code in object, writing an elliptic solver consistent with the compact schemes used for the computation of the convective term was judged unfeasible within the three years deadline, for the reasons explained in Section 6.1.1. As a consequence, the code development of a standard second order solver was undertaken by a post graduate, and conducted, as far as possible, under the mild supervision of the author, who was, at that time, struggling to climb up the steep learning curve of both Fortran programming and MPI at the same time.

The problematic Laplace operator

The Poisson equation for the pressure, concisely written as $\nabla^2 p = \nabla \cdot \mathbf{V}$, hides the consistency between divergence, gradient, and Laplace operators which is automatically satisfied in the continuous space, $\nabla \cdot \nabla = \nabla^2$.

Since the same property does not hold, in general, in the discrete space, it must be enforced by explicitly requiring that the discrete Laplace operator \mathbf{L} be the product of the discrete divergence operator \mathbf{D} times the discrete gradient operator \mathbf{G} , i.e., that $\mathbf{L} = \mathbf{D}\mathbf{G}$, which is not an easy task if compact schemes are concerned. Indeed, if only explicit finite differences are used, the operators \mathbf{D} and \mathbf{G} are sparse, together with their product, the Laplace discrete operator. In turn, solving the elliptic equation means solving a sparse linear system of equations, whose matrix of the coefficients is easily stored thanks to its sparsity, by the way. When compact schemes are used to discretize one-dimensional difference operators, then discrete divergence and gradient operators are not as sparse, and their product is even less sparse.

Indeed, assuming for simplicity an $n \times n$ two-dimensional uniform periodic domain, the generic compact scheme for the first derivative $\mathbf{A}\mathbf{y}' = \mathbf{B}\mathbf{y}$, corresponds to a difference one-dimensional operator which can be symbolically written as $\mathbf{A}^{-1}\mathbf{B}$ and used along both directions, being \mathbf{A} and \mathbf{B} both narrow-banded cyclic matrices, the former symmetric, the latter self-adjoint. The two-dimensional operators for the first derivatives along the two directions can be written as $\mathbf{I} \otimes (\mathbf{A}^{-1}\mathbf{B})$ and $(\mathbf{A}^{-1}\mathbf{B}) \otimes \mathbf{I}$ respectively, where \otimes is the Kronecker product, so that the divergence operator is the 1×2 block matrix

$$\mathbf{D} = [\mathbf{I} \otimes (\mathbf{A}^{-1}\mathbf{B}) \quad (\mathbf{A}^{-1}\mathbf{B}) \otimes \mathbf{I}]$$

whereas the gradient operator is the following 2×1 block matrix, adjoint of \mathbf{D} ,

$$\mathbf{G} = \begin{bmatrix} \mathbf{I} \otimes (\mathbf{A}^{-1}\mathbf{B}) \\ (\mathbf{A}^{-1}\mathbf{B}) \otimes \mathbf{I} \end{bmatrix} = -\mathbf{D}^T$$

Finally, the Laplace operator is equal to their product,

$$\mathbf{L} = \mathbf{D}\mathbf{G} = [\mathbf{I} \otimes (\mathbf{A}^{-1}\mathbf{B}\mathbf{A}^{-1}\mathbf{B}) + (\mathbf{A}^{-1}\mathbf{B}\mathbf{A}^{-1}\mathbf{B}) \otimes \mathbf{I}]$$

whose first addend is a sparse block matrix with *full* blocks, and the second addend is a permuted version of the former.

The problem with \mathbf{L} is that it must not be applied — which is done through the solution of several linear systems —, but computed, since it is the matrix of the coefficients of the elliptic equation that must be solved for \mathbf{p} ,

$$\mathbf{L}\mathbf{p} = \mathbf{D}\mathbf{u}^* \quad (6.1)$$

Since \mathbf{L} is not properly sparse, and its three-dimensional version is even bigger, storing it once and for all, and solving the corresponding system of equations at each time step is a very memory demanding and computationally intensive task, and this approach is definitely infrequent, the only attempt to the author's knowledge being that of Boersma [34].

Other opportunities

An interesting approach, on the base of which the code in object could be improved, was presented by Reis et al. [42], and consists in decomposing Eq. (6.1) in a system of one-dimensional first-order differential equations, and a purely algebraic equation. This system is claimed to be easily solved.

If a more tested approach was desired, then the technique of solving the Poisson equation in the Fourier space, regardless of the boundary conditions on the velocity, used by Laizet et al. [9], should be considered, and deeply investigated. A foreseeable problem in the implementation of such a technique in the source code is that the former really matches the pencil-transposition-based two-dimensional domain decomposition [11], whereas the latter is deeply anchored to the three-dimensional domain decomposition, as is apparent throughout this thesis. On the other hand, since moving a differential equation to the Fourier space roughly means that the differential operators are algebraized, the implementation will possibly only require the parallelization of a matrix product, much like it is already done for any compact-related \mathbf{B} matrix in the present code.

6.1.2 Spectral methods along periodic directions

Spectral solution along periodic directions should be implemented to take advantage of fast Fourier transform algorithm as much as possible.

6.1.3 On time integration

Adaptive Δt on non-uniform grids

A uniform spatial mesh-compatible adaptive time-stepping procedure is already implemented in the code, and activated whenever the input line `CFL_`*(CFL number)* is provided; a constant Δt can be forced by giving in input the line `deltat_`*(Δt)* instead.

In order to adapt the existing procedure to the case of non-uniform spatial meshes, the CFL number relative to each cell should be computed, and the Δt reduced or increased accordingly. Since the local CFL number depends on the local values of both velocity and grid spacing, properly defining it on a non-uniform space grid in a domain decomposition framework is not straightforward. Furthermore, the chunks of code relative to grid generation and handling are not shipped with appropriate functions to handle operations acting *on* the grids, since such tool are not needed for a simulation to be performed. Indeed, the spatial grids are basically three arrays, used as input for the computation of the finite difference coefficients, and then no more. A possible course of action could consist in the implementation of proper grid-handling routines in order to allow the computation of the local CFL number, or, more likely, an approximation.

Approximate projection and multi-step integrators

When the Runge-Kutta time integration method is used, the solver already implements a procedure of zero degree approximate projection for the pressure [90, FSa] (on request, by appending `_approx_proj` to the time integrator identifier in the input file). This reduces the number of Poisson equations for time step from 4 to 1, but degrades the solution. The higher order method shown in [90, FSb] can be implemented to save computations at a lower inaccuracy cost.

Multi-step time integrators, such as those of the Adams-Bashforth family, can be easily implemented, as well, and used as long as a constant time-marching procedure is chosen. This methods would require only one Poisson equation to be solved for time-step, with the almost irrelevant complication of requiring a starting procedure, which would simply consist in using a few Adams-Bashforth methods in sequence, from the first order up to the desired one.

6.1.4 LES

The implementation of sub-grid models is essential to LES runs, and its implementation consists in the addition of a module making use of already existing tools for the computation of the sub-grid term in the filtered Navier-Stokes equations.

6.2 Numerics-related

6.2.1 LU decomposition for tridiagonal matrices

At the time of writing, each time the Thomas algorithm is invoked, it solves the system through a first sweep elimination of the lower diagonal, followed by a backward substitution. *De facto*, it computes the LU decomposition of \mathbf{A} on the run and drops it afterward.

A highly suggested edit to the code is the following. The first time a tridiagonal system is solved, the LU decomposition of the coefficient matrix should permanently overwrite the matrix itself, and a new **LOGICAL** member of the `CDS` datatype, for instance named `isLU`, should be set to `.TRUE.`, so that subsequent calls to the Thomas algorithm will consist of the backward substitution only, thus saving $3(n - 1)$ flops [57].

6.2.2 SPIKE algorithm's reduced system

In the context of the SPIKE algorithm, the coupling matrix of the so-called reduced system, presented as a pentadiagonal matrix in Fig. 3.5 of Section 3.1.2 (see also [18]), can actually be cast as a tridiagonal matrix [19, 22], thus allowing a slightly faster resolution of the reduced system, to the benefit of performance, at the only cost of a marginal edit to the source code, consisting in a reordering of ψ and ψ^0 in Eq. (3.18).

As a side note, it is urgent and straightforward to short-circuit the application of the SPIKE algorithm, when the \mathbf{A} matrix is diagonal, which is trivially the case for explicit finite difference schemes.

6.2.3 Coefficients of compact schemes

The coefficients relative to the compact schemes are computed at runtime in floating point format, through the routine in Listing A.1, and are not hard-coded, so that the program naturally deals with non-uniform grids.

The precision with which these coefficients are computed could represent a problem when extreme precision is needed.

In the case of uniform grids, it should be possible to implement a MATLAB preprocessing script which uses the symbolic tool, as in Listing A.2, to compute fractional representation of the coefficients and write them to file, so that the Fortran code can read them in. Similarly, another possibility is that Fortran uses system calls (through the `SYSTEM` subroutine/function) to compute coefficients through the widely available `calc` Linux tool.

6.3 Programming-related

6.3.1 Serial and/or two dimensional run

At the time of writing, one major flaw of the program is that it cannot be run with less than 1 process per direction, which means that at least $2^3 = 8$ processes are required for the program to successfully start. As a consequence, a serial run cannot be performed, neither can it be run on a one- or two-dimensional MPI Cartesian grid. As side effect, a two-dimensional simulation is not possible, since it would require the use of a single cell along one direction, which would of course belong to one process only.¹

This limitation stems from all code sections whose dependence on the Cartesian grid of processes is hard-coded, mainly those operator and grid-management declarations and subroutines (executed before the actual time integration), which are built upon the widely deprecated master-slave logic. Startup-critical calls should be bypassed along directions with one process along them. Most importantly, strong assumptions are made in the pressure module, which should already undergo massive edits for other reasons explained in Section 6.1.1.

6.3.2 Pointers to avoid temporary arrays

Many procedure calls lead to the copy of large array in memory, due to the use of expressions, instead of variables, as arguments of the calls:

¹Actually the diffusive and convective terms involving the third dimension could be switched off, based on a flag in input, to force a two-dimensional solution, still using 2 processes along the third direction (with the minimum possible number of cells). In the author's opinion, however, it would be not worth it.

```

CALL mysub(var)           ! var is a variable → not copied
CALL mysub(var(2:10))    ! var(2:10) is a expression → copied
                           ! to a temporary array

```

The use of pointers could be a possible opportunity to reduce the overhead due to the creation of temporary arrays, by removing the necessity for these temporary copies.

6.3.3 CDS by rows and JDS formats for banded matrices

As described in Section 4.8.2, the `CDS` datatype is used to store all banded matrices in the Compressed Diagonal Storage format, both **A** and **B** of the compact schemes.

At the time the datatype was coded, the diagonals were mistakenly decided to be stored along the *columns* of the `matrix` member of `CDS`; for instance the j^{th} diagonal of **B** is stored in `(B var)%matrix(:,j)`. When **B** undergoes matrix-vector multiplications, the i^{th} element of the resulting array is computed as the matrix product between a small portion of the given column array, times the in-band portion of the of **B**'s i^{th} row, which is stored in the row `(B var)%matrix(i,:)`, whose elements are *not* consecutive in memory, since Fortran stores matrices in a column major fashion, unlike C++, which uses the row major storage.

So a mandatory edit is to move from the current row-oriented implementation of the `CDS` format, which fits with C++ more then with Fortran, to a column-oriented implementation (the j^{th} diagonal of **B** stored in `(B var)%matrix(j,:)`), in order to save several percent points on the wall clock time elapsed for each multiplication.²

This kind of storage is the most space-efficient, as long as the matrices have a constant bandwidth, which is certainly the case when a periodic direction is concerned; otherwise several zeros are stored, thus wasting memory, as is the case for non-periodic directions. Since the bandwidth of compact schemes' matrices vary only close to the boundaries, and since each matrix is scattered among several processes, it is clear that pro-

²The edit was actually attempted already, and the comparison was performed in a standalone, non-parallel Fortran program, and resulted in an unexpectedly significant reduction of the wall clock time needed for a matrix-vector product. The edit, however, was not pulled in CFD code, since it would have meant substantial a rewrite of the procedures responsible for distribution of the compact matrices among processes, which is a fairly delicate task.

cesses not adjacent to the boundary do not suffer from this problem, since the outer diagonals are identically zero, in the inner part, and can simply be dropped, as a beneficial consequence. The processes touching the boundaries, on the contrary, should store those zeros, which can represent up to half the content of the `<matrix>%matrix` member of a CDS-stored banded matrix; what is more, those zeros will uselessly undergo all scalar multiplications arising from matrix-vector products, thus increasing the computational cost suffered by near-boundary processes.

One suggested solution is to implement the JDS format, more space-efficient for non-constant bandwidth matrices, as defined in [64]. This format should be used only along non-periodic directions, and only for those processes who touch the minus or plus boundaries.

6.3.4 Condensation of MPI calls

The possible advantage in abandoning/changing the master-slave logic intrinsic to the SPIKE algorithm should be investigated.

Considering only those processes belonging to a single pencil within the 3D Cartesian MPI topology, at the current state the first process in the pencil plays the master, and uses `MPI_GATHER` to gather two layers of data from each remaining process, in order to build the array ψ^0 of Eq. (3.18), then solves the reduced system for ψ , and finally scatters the result back to the processes through a call to `MPI_SCATTER`. Within this strategy, each process sends two layers of data to the master, then waits for the master, and finally receives back two new layers, whereas the master has to receive $2 \times n$ layers, solve the system, then send the $2 \times n$ new layers back. It is worthwhile to underline that all first processes along the pencil are master to their pencil. As a consequence, in a $n \times n \times n$ MPI Cartesian grid, there are $3n - 2$ processes which are masters to two pencils, and one process (the process 0) which is master to three pencils. This is a possible reason for poor performance.

An alternative is a call to `MPI_ALLGATHER`, which results in all processes performing $2 \times n$ sends and $2 \times n$ receives, so that each of them can solve the reduced system on its own and proceed. With this hypothetical strategy, all processes would do the same work as only the master process does in the strategy currently implemented, with apparently no gain. Actually the author believes that a single call to `MPI_ALLGATHER`, in place of the twin calls to `MPI_GATHER`/`MPI_SCATTER`, gives an opportunity to the compiler for an optimization of the communication pattern.

The subroutine `MPI_TYPE_CREATE_STRUCT` could be used to group in one datatype the three datatypes used by `MPI_NEIGHBOR_ALLTOALLW` to exchange each component of velocity, thus allowing a single call to exchange all the near-boundary flowfield between neighboring processes, hopefully to the benefit of the compiler's optimization capabilities. Indeed, it is common knowledge that reducing the number of messages likely improves the performance of an MPI program [88].

6.3.5 Fortran-native vectorization and OpenMP-MPI

MPI parallelization was accomplished by domain decomposition and explicit message passing. Explicit data exchange at inter-process boundaries takes place before every interpolation/differentiation. After the exchange, each process has to perform repetitive and independent operations along all the pencils it handles, and this is coded in two nested `DO` loops. On vector parallel mainframe, high performance can be achieved with the vectorization (i.e., linearization) of these loops inside a sub-domain.

When possible, the `PURE` and `ELEMENTAL` attributes should be added to procedures (both `FUNCTIONS` and `SUBROUTINES`). Plain `DO` loops should be converted to `FORALL` constructs/statements; likewise, `DO` loops enclosing conditionals constructs, such as `IFs` and `SELECT CASEs`, should be converted to `WHERE` constructs/statements. These edits, would likely result in sensible improvements in performance [63], especially if associated to a proprietary, well-optimizing Fortran compiler, since they could thus take advantage of vectorization capabilities of modern hardware (especially GPUs).

In order to parallelize each of the aforementioned loops OpenMP directives can be used, as well. With respect to the Fortran-native approach, they would leave the Fortran code untouched, since the directive are simply Fortran comments, which are interpreted as directives for the library.

6.3.6 Availability

The code has been tested only against `gfortran`, the Fortran compiler shipped with the GNU Compiler Collection, and the OpenMPI implementation of MPI, the reason being that the workstation the author was provided with, and on which he have conducted the development, was equipped with these compilers/libraries by the author himself. As far as the author had the opportunity to perform some check on CINECA's Marconi HPC facility, the latest version of the code is still non-compilable

with Intel Fortran, due to the well-circumscribed use some non-standard-conforming Fortran and MPI expressions, which are unexpectedly interpreted correctly by `gfortran/mpifort` and not by `ifort/mpiifort`.

The mandatory course of action for the Intel compilers and library to be used, is to perform some troubleshooting to identify the specific lines of code responsible for the incompatibility and change them accordingly.

Optionally, the `Makefile` should be upgraded to automatically handle multiple compilers.

Appendix A

Functions to compute finite differences' coefficients

This appendix contains the `function` that applies the method developed in Section 2.6 for the determination of the coefficients of a general asymmetric scheme. This routine has served to populate finite difference matrix operators.

The codes provides a significant freedom in the choice of the stencil corresponding to the desired scheme:

- the linear scheme can contain virtually *any* number of nodes both in the RHS and in the LHS,
- each of these nodes can be virtually *anyhow* distributed on the real axis,
- a derivative of virtually *any* order of differentiation can be located in each of the nodes,

where the word “virtually” refers to the fact that too wide stencils and/or too high orders of differentiation could require too onerous symbolic calculations, which are preferred to floating point operations for generating fractional coefficients. It is worth to mention that no such problems has been encountered in the development of the libraries of compact schemes.

In the following, both the used Fortran, and the more more succinct MATLAB versions are listed, in Listings A.1 and A.2 respectively.

Listing A.1: Fortran version

```

FUNCTION cmp_coeff(LHS_s, RHS_s, LHS_o, RHS_o, c) RESULT(coeffs)
! This subroutine determines the coefficients of a scheme
! collocated in c and with the left hand side stencil, where
! derivatives of order LHS_o are unknown, defined by LHS_s, and
! with the right hand side stencil, where derivatives of order
! RHS_o are known, defined by RHS_s. The output is in the array
! coeffs. For instance, the following call
!
!   coeffs = cmp_coeff([0.0,1.0],[-.5,0.0,1.0,2.0,3.0], &
!                     & [2,2],[1,0,0,0,0],0.0)
!
! Gives the coefficients of the compact scheme whose stencil is
! the following
!
!   coeffs(2)  coeffs(3)      coeffs(4)      coeffs(5)
!   coeffs(6)
!
!   \          o          o          o          o
!   |-----|-----|-----|-----|
!   |          |          |          |
!   |          ||         ||
!   |          1        coeffs(1)
!
!                                     IV ordine
!
! TODO: LHS_s and RHS_s should be joined in one INTENT(IN)
! variable (the same should be done for LHS_o and RHS_o); then
! the CALLER should arrange for the sign of RHS' coefficients to
! be changed. This edit would (1) slim the FUNCTION, and (2) be
! more coherent with the fact a FD scheme can be thought of as a
! linear combination of several Taylor series (one for each term
! of the FD) which sum up to a quantity which is zero up to a
! given order.

IMPLICIT NONE

! input/output variables
REAL,    DIMENSION(:),                INTENT(IN) :: LHS_s, RHS_s
INTEGER, DIMENSION(SIZE(LHS_s)),    INTENT(IN) :: LHS_o
INTEGER, DIMENSION(SIZE(RHS_s)),    INTENT(IN) :: RHS_o
REAL,    INTENT(IN) :: c
REAL,    DIMENSION(SIZE([LHS_s, RHS_s]) - 1) :: coeffs

! internal variables
INTEGER :: ncoeff, ic, sz, k, i, kk, info
REAL    :: xmin, xc
REAL,    DIMENSION(SIZE(LHS_s)) :: xL
REAL,    DIMENSION(SIZE(RHS_s)) :: xR
INTEGER, DIMENSION(SIZE([LHS_s, RHS_s])) :: d

```

```

REAL,      DIMENSION(SIZE([LHS_s, RHS_s])) :: x
REAL,      DIMENSION(SIZE([LHS_s, RHS_s])) :: tmp
REAL,      DIMENSION(SIZE([LHS_s, RHS_s]) - 1) :: ipiv
REAL,      DIMENSION(SIZE([LHS_s, RHS_s]) - 1, &
                        & SIZE([LHS_s, RHS_s]) - 1) :: A

sz = SIZE([LHS_s, RHS_s])
ncoeff = sz - 1

xmin = MIN(MINVAL(LHS_s), MINVAL(RHS_s))
xL(:) = LHS_s - xmin + 1
xR(:) = RHS_s - xmin + 1
x(:)  = [xL, xR]
d(:)  = [LHS_o, RHS_o]
xc    = c - xmin + 1
ic    = first(find(x == xc))

DO k = 1, ncoeff
    tmp(:) = [(PRODUCT([(kk, kk = k - 1, k - d(i), -1)]), &
              & i = 1, sz)]*(x**(k - d - 1))
    coeffs(k) = - tmp(ic)
    A(k,:)    = tmp([(kk, kk = 1, ic - 1), &
                    & (kk, kk = ic + 1, sz)])
END DO

CALL DGESV(ncoeff, 1, A, ncoeff, ipiv, coeffs, ncoeff, info)

! change sign for coefficients on the RHS
coeffs(SIZE(xL):) = - coeffs(SIZE(xL):)

END FUNCTION cmp_coeff

```

Listing A.2: MATLAB version

```

function [coeff,A,b] = coefficients(LHS_stencil,RHS_stencil,...
                                   LHS_degrees,RHS_degrees,colloc)
%COEFFICIENTS   Coefficients of a numeric scheme.
% This function determines the coefficients of a numerical
% scheme % collocated in colloc and which locating on the
% points defined in the array LHS_stencil the unknown
% derivatives of order defined in the array LHS_degrees;
% similarly it locates on the points defined by the array
% RHS_stencil the derivatives of order defined by the array
% RHS_degrees.
% !!!ATTENTION!!! The collocation point of the scheme must
% be one of the points of the array LHS_stencil.

```

```

%
% As an example the following command
%
% coeff = COEFFICIENTS([0 1],[-.5 0 1 2 3], ...
%                    [2 2],[1 0 0 0 0],0)
%
% gives in output the coefficients of the following scheme
%
%      \      o      o      o      o
%      |_____|_____|_____|_____|
%      |      |      |      |
%      1\\      \\
%
%                                     IV order
%
% from left to right and from bottom to top.
%
% Negative powers of zero are obviously not well defined, so the
% terse writing 0^k cannot be used in place of kron_(0,k).
% We simply shift every abscissa to the positive semi-axis.
xmin = min([LHS_stencil RHS_stencil]); % minimum abscissa
xL = LHS_stencil - xmin + 1;           % shift to x > 0
xR = RHS_stencil - xmin + 1;
xc = colloc - xmin + 1;

dL = LHS_degrees;
dR = RHS_degrees;

ncoeff = length(xL) + length(xR) - 1; % how many coefficients
ic = find(xL == xc);                  % index of colloc. point

A = sym('A',[ncoeff ncoeff+1]);

x = [xL xR];
d = [dL dR];
temp = zeros(size(d));
% build up the system
for k = 0:ncoeff-1
    for ii = 1:length(d)
        temp(ii) = prod(k:-1:k-d(ii)+1);
    end
    temp(1:length(xL)) = -temp(1:length(xL));
    A(k+1,:) = (x.^(k - d)).*temp;
end

b = - A(:,ic);
A(:,ic) = [];

coeff = A\b;

```

Bibliography

- [1] Christopher J GREENSHIELDS. “OpenFOAM user guide”. *OpenFOAM Foundation Ltd, version 3.1* (2015).
- [2] Hrvoje JASAK. “OpenFOAM: open source CFD in research and industry”. *International Journal of Naval Architecture and Ocean Engineering* 1.2 (2009), pp. 89–94.
- [3] Hrvoje JASAK, Aleksandar JEMCOV, Zeljko TUKOVIC, et al. “OpenFOAM: A C++ library for complex physics simulations”. *International workshop on coupled methods in numerical dynamics*. Vol. 1000. IUC Dubrovnik, Croatia. 2007, pp. 1–20.
- [4] Francisco PALACIOS, Thomas D ECONOMON, Aniket ARANAKE, Sean R COPELAND, Amrita K LONKAR, Trent W LUKACZYK, David E MANOSALVAS, Kedar R NAIK, Santiago PADRON, Brendan TRACEY, et al. “Stanford university unstructured (SU^2): Analysis and design technology for turbulent flows”. *52nd Aerospace Sciences Meeting*. 2014, p. 0243.
- [5] Francisco PALACIOS, Juan ALONSO, Karthikeyan DURAISAMY, Michael COLONNO, Jason HICKEN, Aniket ARANAKE, Alejandro CAMPOS, Sean COPELAND, Thomas ECONOMON, Amrita LONKAR, et al. “Stanford University Unstructured (SU^2): an open-source integrated computational environment for multi-physics simulation and design”. *51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*. 2013, p. 287.
- [6] Domenico VERBICARO. “OpenFOAM and SU^2 open-source CFD codes: benchmark and validation for aerodynamic applications”. Master Thesis. Università degli Studi di Napoli “Federico II”, 2016.

- [7] Jing GONG, Stefano MARKIDIS, Michael SCHLIEPHAKE, Erwin LAURE, Dan HENNINGSON, Philipp SCHLATTER, Adam PEPLINSKI, Alistair HART, Jens DOLESCHAL, David HENTY, et al. “Nek5000 with OpenACC”. *International Conference on Exascale Applications and Software*. Springer. 2014, pp. 57–68.
- [8] Adam PEPLINSKI, Philipp SCHLATTER, PF FISCHER, and Dan S HENNINGSON. “Stability tools for the spectral-element code Nek5000: Application to Jet-in-Crossflow”. *Spectral and High Order Methods for Partial Differential Equations-ICOSAHOM 2012*. Springer, 2014, pp. 349–359.
- [9] Sylvain LAIZET and Ning LI. “Incompact3d: A powerful tool to tackle turbulence problems with up to $\mathcal{O}(10^5)$ computational cores”. *International Journal for Numerical Methods in Fluids* 67.11 (2011), pp. 1735–1757.
- [10] Sylvain LAIZET, Eric LAMBALLAIS, and John Christos VASSILICOS. “A numerical strategy to combine high-order schemes, complex geometry and parallel computing for high resolution DNS of fractal generated turbulence”. *Computers & Fluids* 39.3 (2010), pp. 471–484.
- [11] Ning LI and Sylvain LAIZET. “2DECOMP & FFT-a highly scalable 2d decomposition library and FFT interface”. *Cray User Group 2010 conference*. 2010, pp. 1–13.
- [12] Karla MORRIS, Damian ROUSON, and Salvatore FILIPPONE. *Parallel Programming in Modern Fortran*. Nov. 2014. URL: http://sc14.supercomputing.org/schedule/event_detail-evid=tut112.html.
- [13] John LEVESQUE. *Fortran Is 60 Years Old - Has It Changed for the Better?* Nov. 2017. URL: <https://sc17.supercomputing.org/presentation/?id=bof202&sess=sess329>.
- [14] Sanjiva K. LELE. “Compact finite difference schemes with spectral-like resolution”. *Journal of Computational Physics* 103.1 (1992), pp. 16–42.
- [15] Jae Wook KIM and Richard D. SANDBERG. “Efficient parallel computing with a compact finite difference scheme”. *Computers & Fluids* 58 (2012), pp. 70–87.

- [16] Alex POVITSKY and Philip J. MORRIS. “A higher-order compact method in space and time based on parallel implementation of the Thomas algorithm”. *Journal of Computational Physics* 161.1 (2000), pp. 182–203.
- [17] Eric POLIZZI and Ahmed H. SAMEH. “A parallel hybrid banded system solver: the SPIKE algorithm”. *Parallel Computing* 32.2 (2006), pp. 177–194.
- [18] Yingchong SITU, Chandra S. MARTHA, Matthew E. LOUIS, Zhiyuan LI, Ahmed H. SAMEH, Gregory A. BLAISDELL, and Anastasios S. LYRINTZIS. “Petascale large eddy simulation of jet engine noise based on the truncated SPIKE algorithm”. *Parallel Computing* 40.9 (2014), pp. 496–511.
- [19] Debojyoti GHOSH, Emil M. CONSTANTINESCU, and Jed BROWN. “Efficient implementation of nonlinear compact schemes on massively parallel platforms”. *SIAM Journal on Scientific Computing* 37.3 (2015), pp. C354–C383.
- [20] Stéphane ABIDE and Stéphane VIAZZO. “A 2D compact fourth-order projection decomposition method”. *Journal of Computational Physics* 206.1 (2005), pp. 252–276.
- [21] Jeffrey Mark MC NALLY, Lawrence E. GAREY, and Ruth E. SHAW. “A split-correct parallel algorithm for solving tridiagonal symmetric Toeplitz systems”. *International Journal of Computer Mathematics* 75.3 (2000), pp. 303–313.
- [22] Nathan MATTOR, Timothy J. WILLIAMS, and Dennis W. HEWETT. “Algorithm for solving tridiagonal matrix problems in parallel”. *Parallel Computing* 21.11 (1995), pp. 1769–1782.
- [23] Pierluigi AMODIO and Luigi BRUGNANO. “Parallel factorizations and parallel solvers for tridiagonal linear systems”. *Linear algebra and its applications* 172 (1992), pp. 347–364.
- [24] Ronnie KNIKKER. “Study of a staggered fourth-order compact scheme for unsteady incompressible viscous flows”. *International journal for numerical methods in fluids* 59.10 (2009), pp. 1063–1092.
- [25] Santhanam NAGARAJAN, Sanjiva K. LELE, and Joel H. FERZIGER. “A robust high-order compact method for large eddy simulation”. *Journal of Computational Physics* 191.2 (2003), pp. 392–419.

- [26] Xuliang LIU, Shuhai ZHANG, Hanxin ZHANG, and Chi-Wang SHU. “A new class of central compact schemes with spectral-like resolution I: Linear schemes”. *Journal of Computational Physics* 248 (2013), pp. 235–256.
- [27] Matteo BERNARDINI, Sergio PIROZZOLI, and Paolo ORLANDI. “Velocity statistics in turbulent channel flow up to $Re_\tau = 4000$ ”. *Journal of Fluid Mechanics* 742 (2014), pp. 171–191.
- [28] James W. DEARDORFF. “A numerical study of three-dimensional turbulent channel flow at large Reynolds numbers”. *Journal of Fluid Mechanics* 41.2 (1970), pp. 453–480.
- [29] U. K. N. G. GHIA, Kirti N. GHIA, and C. T. SHIN. “High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method”. *Journal of Computational Physics* 48.3 (1982), pp. 387–411.
- [30] Francis H. HARLOW and J. Eddie WELCH. “Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface”. *Physics of Fluids* 8.12 (1965), pp. 2182–2189.
- [31] Zhenfu TIAN, Xian LIANG, and Peixiang YU. “A higher order compact finite difference algorithm for solving the incompressible Navier–Stokes equations”. *International Journal for Numerical Methods in Engineering* 88.6 (2011), pp. 511–532.
- [32] Peter C. CHU and Chenwu FAN. “A three-point combined compact difference scheme”. *Journal of Computational Physics* 140.2 (1998), pp. 370–399.
- [33] Bendiks Jan BOERSMA. “A staggered compact finite difference formulation for the compressible Navier–Stokes equations”. *Journal of Computational Physics* 208.2 (2005), pp. 675–690.
- [34] Bendiks Jan BOERSMA. “A 6th order staggered compact finite difference method for the incompressible Navier–Stokes and scalar transport equations”. *Journal of Computational Physics* 230.12 (2011), pp. 4940–4954.
- [35] Knut AKSELVOLL and Parviz MOIN. “Large-eddy simulation of turbulent confined coannular jets”. *Journal of Fluid Mechanics* 315 (1996), pp. 387–411.

- [36] Charles D. PIERCE and Parviz MOIN. “Progress-variable approach for large-eddy simulation of non-premixed turbulent combustion”. *Journal of Fluid Mechanics* 504 (2004), pp. 73–97.
- [37] Yohei MORINISHI, Thomas S. LUND, Oleg V. VASILYEV, and Parviz MOIN. “Fully conservative higher order finite difference schemes for incompressible flow”. *Journal of Computational Physics* 143.1 (1998), pp. 90–124.
- [38] Dimitris DRIKAKIS, Marco HAHN, Andrew MOSEDALE, and Ben THORNBUR. “Large eddy simulation using high-resolution and high-order methods”. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 367.1899 (2009), pp. 2985–2997.
- [39] George S. CONSTANTINESCU and Sanjiva K. LELE. “Large eddy simulation of a near sonic turbulent jet and its radiated noise”. *39th Aerospace Sciences Meeting and Exhibit*. 2001, p. 376.
- [40] Parviz MOIN, Kyle SQUIRES, William H. CABOT, and Sangsan LEE. “A dynamic subgrid-scale model for compressible turbulence and scalar transport”. *Physics of Fluids A: Fluid Dynamics* 3.11 (1991), pp. 2746–2757.
- [41] Arthur G. KRAVCHENKO and Parviz MOIN. “On the effect of numerical errors in large eddy simulations of turbulent flows”. *Journal of Computational Physics* 131.2 (1997), pp. 310–322.
- [42] Gabriela Aparecida dos REIS, Italo Valença Mariotti TASSO, Leandro Franco de SOUZA, and José Alberto CUMINATO. “A compact finite differences exact projection method for the Navier–Stokes equations on a staggered grid with fourth-order spatial precision”. *Computers & Fluids* 118 (2015), pp. 19–31.
- [43] Tapan K. SENGUPTA, Anurag DIPANKAR, and Anupindi Kameswara RAO. “A new compact scheme for parallel computing using domain decomposition”. *Journal of Computational Physics* 220.2 (2007), pp. 654–677.
- [44] Parviz MOIN. *Fundamentals of Engineering Numerical Analysis, 2nd Edition*. Cambridge University Press, 2010.

- [45] Alfio QUARTERONI, Fausto SALERI, and Paola GERVASIO. *Scientific Computing with MATLAB and Octave, 3rd Edition*. Ed. by Timothy J. BARTH, Michael GRIEBEL, David E. KEYES, Risto M. NIEMINEN, Dirk ROOSE, and Tamar SCHLICK. Vol. 2. Texts in Computational Science and Engineering. Berlin: Springer, 2010.
- [46] David GOTTLIEB and Steven A. ORSZAG. *Numerical Analysis of Spectral Methods: theory and applications*. Vol. 26. CBMS-NSF Regional Conference Series in Applied Mathematics. Philadelphia: SIAM, 1977.
- [47] Claudio CANUTO, M. Yosuff HUSSAINI, Alfio QUARTERONI, and Thomas A. ZANG. *Spectral Methods in Fluid Dynamics*. New York: Springer, 1988.
- [48] Robert S. ROGALLO and Parviz MOIN. “Numerical simulation of turbulent flows”. *Annual review of fluid mechanics* 16.1 (1984), pp. 99–137.
- [49] John KIM, Parviz MOIN, and Robert MOSER. “Turbulence statistics in fully developed channel flow at low Reynolds number”. *Journal of Fluid Mechanics* 177 (1987), pp. 133–166.
- [50] Philippe R. SPALART. “Direct simulation of a turbulent boundary layer up to $Re_\theta = 1410$ ”. *Journal of Fluid Mechanics* 187 (1988), pp. 61–98.
- [51] Enrico Maria DE ANGELIS, Gennaro COPPOLA, Francesco CAPUANO, and Luigi de LUCA. “Derivation of New Staggered Compact Schemes with Application to Navier–Stokes Equations”. *Applied Sciences* 8.7 (2018), p. 1066.
- [52] Gennaro COPPOLA. “La teoria della Local Matched Reconstruction e le sue applicazioni in fluidodinamica numerica”. Ph.D. thesis. Università degli Studi di Napoli “Federico II”, Nov. 2001.
- [53] Enrico Maria DE ANGELIS. “Parallel implementation of compact schemes for Navier–Stokes simulations”. Master Thesis. Università degli Studi di Napoli “Federico II”, 2015.
- [54] Bengt FORNBERG. “Generation of finite difference formulas on arbitrarily spaced grids”. *Mathematics of Computation* 51.184 (Oct. 1988), pp. 699–706.

- [55] Blair SWARTZ and Burton WENDROFF. “The comparative efficiency of certain finite element and finite difference methods for a hyperbolic problem”. *Conference on the Numerical Solution of Differential Equations*. Vol. 363. 1974, pp. 153–163.
- [56] Philip G. DRAZIN. *Introduction to Hydrodynamic Stability*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2002. ISBN: 9780521009652.
- [57] Alfio QUARTERONI, Riccardo SACCO, Fausto SALERI, and Paola GERVASIO. *Matematica Numerica*. UNITEXT. Springer Milan, 2014. ISBN: 9788847056442. URL: <https://books.google.it/books?id=RSHpCAAAQBAJ>.
- [58] Stephen DEMKO, William F. MOSS, and Philip W. SMITH. “Decay Rates for Inverses of Band Matrices”. *Mathematics of Computation* 43.168 (Oct. 1984), pp. 491–499.
- [59] Francesco CAPUANO, Andrea MASTELLONE, and Enrico Maria DE ANGELIS. “A conservative overlap method for multi-block parallelization of compact finite-volume schemes”. *Computers & Fluids* 159 (2017), pp. 327–337.
- [60] Jon LOELIGER and Matthew MCCULLOUGH. *Version Control with Git: Powerful tools and techniques for collaborative software development*. " O'Reilly Media, Inc.", 2012.
- [61] Emma Jane Hogbin WESTBY. *Git for teams: a user-centered approach to creating efficient workflows in Git*. " O'Reilly Media, Inc.", 2015.
- [62] Peter BELL and Brent BEER. *Introducing GitHub: A non-technical guide*. " O'Reilly Media, Inc.", 2014.
- [63] Stephen J. CHAPMAN. *Fortran 95/2003 for Scientists and Engineers*. McGraw-Hill, 2008. ISBN: 9780071285780. URL: <https://books.google.it/books?id=hLkQAAAACAAJ>.
- [64] Richard BARRETT, Michael BERRY, Tony F. CHAN, James DEMMEL, June DONATO, Jack DONGARRA, Victor EIJKHOUT, Roldan POZO, Charles ROMINE, and Henk van der VORST. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. Philadelphia, PA: SIAM, 1994. URL: http://www.netlib.org/linalg/html_templates/report.html.

- [65] Torsten HOEFLER and Jesper Larsson TRAFF. “Sparse collective operations for MPI”. *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*. IEEE. 2009, pp. 1–8.
- [66] Xiyang IA YANG and Rajat MITTAL. “Acceleration of the Jacobi iterative method by factors exceeding 100 using scheduled relaxation”. *Journal of Computational Physics* 274 (2014), pp. 695–708.
- [67] Joel H. FERZIGER and Milovan PERIC. *Computational methods for fluid dynamics*. Springer Science & Business Media, 2012.
- [68] Stefan ALBENSOEDER and Hendrik C. KUHLMANN. “Accurate three-dimensional lid-driven cavity flow”. *Journal of Computational Physics* 206.2 (2005), pp. 536–558.
- [69] Odus R. BURGGRAF. “Analytical and numerical studies of the structure of steady separated flows”. *Journal of Fluid Mechanics* 24.1 (1966), pp. 113–151.
- [70] H. Keith MOFFATT. “Viscous and resistive eddies near a sharp corner”. *Journal of Fluid Mechanics* 18.1 (1964), pp. 1–18.
- [71] Clifford HANCOCK, E. LEWIS, and H. Keith MOFFATT. “Effects of inertia in forced corner flows”. *Journal of Fluid Mechanics* 112 (1981), pp. 315–327.
- [72] Geoffrey I. TAYLOR. *On scraping viscous fluid from a plane surface. Reprinted in The Scientific Papers of Sir Geoffrey Ingram Taylor, vol. IV*. 1962.
- [73] Murli M. GUPTA, Ram P. MANOHAR, and Ben NOBLE. “Nature of viscous flows near sharp corners”. *Computers & Fluids* 9.4 (1981), pp. 379–388.
- [74] Rob SCHREIBER and Herbert B. KELLER. “Driven cavity flows by efficient numerical techniques”. *Journal of Computational Physics* 49.2 (1983), pp. 310–333.
- [75] Olivier BOTELLA. “On the solution of the Navier-Stokes equations using Chebyshev projection schemes with third-order accuracy in time”. *Computers & Fluids* 26.2 (1997), pp. 107–116.
- [76] Shuling HOU, Qisu ZOU, Shiyi CHEN, Gary DOOLEN, and Allen C. COGLEY. “Simulation of cavity flow by the lattice Boltzmann method”. *Journal of Computational Physics* 118.2 (1995), pp. 329–347.

- [77] Charles-Henri BRUNEAU and Claude JOURON. “An efficient scheme for solving steady incompressible Navier-Stokes equations”. *Journal of Computational Physics* 89.2 (1990), pp. 389–413.
- [78] Herman J. H. CLERCX and Gert Jan F. van HEIJST. “Dissipation of kinetic energy in two-dimensional bounded flows”. *Physical Review E* 65.6 (2002), p. 066305.
- [79] Paolo ORLANDI. “Vortex dipole rebound from a wall”. *Physics of Fluids A: Fluid Dynamics (1989-1993)* 2.8 (1990), pp. 1429–1436.
- [80] Katuhiko GODA. “A multistep technique with implicit difference schemes for calculating two-or three-dimensional cavity flows”. *Journal of Computational Physics* 30.1 (1979), pp. 76–95.
- [81] Graham de VAHL DAVIS and Gordon D. D. MALLINSON. “An evaluation of upwind and central difference approximations by a study of recirculating flow”. *Computers & Fluids* 4.1 (1976), pp. 29–43.
- [82] Michel DEVILLE, Thien-Hiep LÊ, and Yves MORCHOISNE. “Numerical simulation of 3-D Incompressible Unsteady Viscous Laminar Flows: The Test Problems”. *Numerical Simulation of 3-D Incompressible Unsteady Viscous Laminar Flows*. Springer, 1992.
- [83] Stefan ALBENSOEDER, Hendrik C. KUHLMANN, and Hans J. RATH. “Three-dimensional centrifugal-flow instabilities in the lid-driven-cavity problem”. *Physics of Fluids* 13.1 (2001), pp. 121–135.
- [84] Marc E. BRACHET, Daniel I. MEIRON, Steven A. ORSZAG, B. G. NICKEL, Rudolf H. MORF, and Uriel FRISCH. “Small-scale structure of the Taylor–Green vortex”. *Journal of Fluid Mechanics* 130 (1983), pp. 411–452.
- [85] Dimitris DRIKAKIS, Christer FUREBY, Fernando F GRINSTEIN, and David YOUNGS. “Simulation of transition and turbulence decay in the Taylor–Green vortex”. *Journal of Turbulence* 8 (2007), N20.
- [86] Marc E. BRACHET. “Direct simulation of three-dimensional turbulence in the Taylor–Green vortex”. *Fluid dynamics research* 8.1-4 (1991), p. 1.
- [87] Wim M. VAN REES, Anthony LEONARD, Dale .I. PULLIN, and Petros KOUMOUTSAKOS. “A comparison of vortex and pseudo-spectral methods for the simulation of periodic vortical flows at high Reynolds numbers”. *Journal of Computational Physics* 230.8 (2011), pp. 2794–2805.

- [88] Peter PACHECO. *An introduction to parallel programming*. Elsevier, 2011.
- [89] Philippe MERCIER and Michel DEVILLE. “A multidimensional compact higher-order scheme for 3-d Poisson’s equation”. *Journal of Computational Physics* 39.2 (1981), pp. 443–455.
- [90] Francesco CAPUANO, Gennaro COPPOLA, Matteo CHIATTO, and Luigi de LUCA. “Approximate projection method for the incompressible Navier–Stokes equations”. *AIAA Journal* 54.7 (2016), pp. 2179–2182.
- [91] Hung LE and Parviz MOIN. “An improvement of fractional step methods for the incompressible Navier-Stokes equations”. *Journal of Computational Physics* 92.2 (1991), pp. 369–379.